Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Matematyczny
specjalność: zastosowania rachunku prawdopodobieństwa i statystyki

Cyryl Karpiński

Zastosowanie metod MCMC do dekodowania zaszyfrowanego tekstu i problemu komiwojażera

Praca magisterska
napisana pod kierunkiem
dra hab. Pawła Lorka
(z poprawkami)

Wrocław 2020

Spis treści

1 Wstęp

W swojej pracy użyję stosunkowo nowego podejścia do dekodowania zaszyfrowanych wiadomości tekstowych za pomocą próbkowania Monte Carlo łańcuchami Markowa. Podejście zostało zastosowane najpierw dla szyfrów podstawieniowych przez Diaconisa (opisane w [1]), rozwinięte przez Connora w [2], a następnie jeszcze rozszerzone na szyfry transpozycyjne przez Chena i Rosenthala w [3]. Moim celem jest zastosowanie podobnego podejścia do ataku na szyfry z rodziny szyfrów polialfabetycznych, gdzie takie podejście – według mojej wiedzy – do tej pory nie było wykorzystywane. Najpierw zaprezentuję stworzony przeze mnie algorytm deterministyczny (a raczej rodzinę algorytmów) rozwiązujący problem w postawionej w tej pracy wersji, następnie pokażę, że da się osiągać szybciej nie gorsze wyniki za pomocą metod MCMC. Postaram się oszacować istotne parametry algorytmów deszyfrujących – złożoność pamięciową i czasową oraz tempo zbieżności (w przypadku algorytmów niedeterministycznych).
Ponadto pokażę, jak użyć metod MCMC do rozwiązania symetrycznego problemu komiwojażera. Przedstawię pewne intuicje związane z tym podejściem, a następnie zaprezentuję działanie algorytmu w praktyce na niektórych instancjach problemu z benchmarkowego zbioru danych wejściowych [4].

Pierwsze rozdziały pracy będą poświęcone wprowadzeniu do teorii łańcuchów Markowa i metod Monte Carlo wraz z dowodami najważniejszych dla tej pracy twierdzeń. W następnej kolejności zajmę się szczegółowym opisem problemu dekodowania i algorytmów zastosowanych w celu jego rozwiązania. Na koniec części dotyczącej dekodowania zaprezentuję wyniki uzyskane przez algorytmy w symulacjach (rozszyfrowywanie zaszyfrowanych fragmentów tekstu w języku angielskim) wraz z ich omówieniem.
Potem przejdę do opisu podejścia MCMC przyjętego do rozwiązania problemu komiwojażera i przedstawię wyniki uzyskane przez algorytmy korzystające z tego podejścia na benchmarkowym zbiorze danych wejściowych.
Na zakończenie krótko omówię program napisany w celu prezentacji zagadnień przedstawionych w tej pracy i podsumuję całość.

Do implementacji algorytmów i symulacji użyłem języka Python3.

2 Łańcuchy Markowa

Przypomnę w tym rozdziale podstawowe zagadnienia z teorii łańcuchów Markowa. Będę zajmował się tylko łańcuchami Markowa (czyli procesami Markowa w czasie dyskretnym) w skończonej przestrzeni stanów – właśnie takie bowiem łańcuchy będą modelowane w celu rozwiązania postawionych problemów. Zajmuję się tutaj pokazaniem teorii łańcuchów Markowa w ogólności, jednak ze szczególnym naciskiem na zachowanie *jednorodnych* łańcuchów Markowa w nieskończoności (zbieżności wg rozkładu, prawa wielkich liczb). Treści przedstawione w tym rozdziale stanowią podstawę teoretyczną dla technik opisanych później.

2.1 Definicje i podstawowe własności

Definicja 2.1.1. *Łańcuchem Markowa (**ŁM**) nazywamy ciąg zmiennych losowych $\boldsymbol{X} = (X_n)_{n\in\{0,1,2...\}}$ – określonych na wspólnej przestrzeni probabilistycznej $\{\Omega, \mathcal{F}, P\}$ przyjmujących wartości z przestrzeni stanów S – mający następującą własność (Markowa):*
Dla $n = 0, 1, 2...$ oraz $j, i_0, i_1, i_2...i_n \in S$ zachodzi:

$$P(X_{n+1} = j|X_n = i_n, X_{n-1} = i_{n-1}, ...X_0 = i_0) = P(X_{n+1} = j|X_n = i_n),$$

jeśli tylko powyższe prawdopodobieństwa warunkowe są dobrze zdefiniowane (to samo 'jeśli' dotyczy wszystkich późniejszych definicji i twierdzeń, w których pojawia się prawdopodobieństwo warunkowe).

Tak jak ustaliliśmy, przyjmujemy $|S| < \infty$. Wartość przyjętą przez X_k będziemy nazywać stanem łańcucha $\boldsymbol{X}$ w k-tym kroku. Z powyższej definicji wynika, że stan w $(k+1)$-szym kroku zależy tylko od stanu w k-tym kroku i nie zależy od stanów we wcześniejszych krokach. Ponieważ przestrzeń stanów jest skończona, więc możemy założyć: $S = \{1, 2, 3, 4...|S|\}$. Często w skrócie zamiast $\boldsymbol{X} = (X_n)_{n\in\{0,1,2...\}}$ będziemy pisać po prostu (X_n).

Fakt 2.1.2. *Niech $0 \le t_0 < t_1 < ... < t_k < n$ oraz $i_0, i_1, ...i_k, i_n, j \in S$. Dla łańcucha Markowa (X_n) zachodzi:*

$$P(X_{n+1} = j|X_n = i_n, X_{t_k} = i_k, ...X_{t_0} = i_0) = P(X_{n+1} = j|X_n = i_n).$$

Dowód. Wystarczy skorzystać ze wzoru na prawdopodobieństwo całkowite: rozbić zbiór $\{X_n = i_n, X_{t_k} = i_k, ...X_{t_0} = i_0\}$ ze względu na wartości X_t dla t, takich że $t < n$ i $t \notin \{t_0, ...t_k\}$. Korzystając następnie z własności Markowa dla każdego elementu takiego rozbicia, dostajemy tezę. □

Lemat 2.1.3. *Dla łańcucha Markowa (X_n), dla $i_0, i_1, ...i_{n+k} \in S$ oraz dla $k > 0$ zachodzi:*

$$P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1}|X_n = i_n, X_{n-1} = i_{n-1}, ...X_0 = i_0) =$$
$$P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1}|X_n = i_n).$$

Dowód. Indukcyjnie po k, dla $k = 1$ jest to po prostu własność Markowa. Załóżmy, że teza zachodzi dla k. Wtedy dla $k + 1$:

$$P(X_{n+k+1} = i_{n+k+1}, ...X_{n+1} = i_{n+1}|X_n = i_n, X_{n-1} = i_{n-1}, ...X_0 = i_0) =$$
$$P(X_{n+k+1} = i_{n+k+1}|X_{n+k} = i_{n+k}, ...X_0 = i_0)P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1}|X_n = i_n, ..., X_0 = i_0) =$$
$$P(X_{n+k+1} = i_{n+k+1}|X_{n+k} = i_{n+k})P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1}|X_n = i_n) =$$
$$P(X_{n+k+1} = i_{n+k+1}|X_{n+k} = i_{n+k}, ..., X_n = i_n)P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1}|X_n = i_n) =$$
$$P(X_{n+k+1} = i_{n+k+1}, ...X_{n+1} = i_{n+1}|X_n = i_n),$$

gdzie skorzystaliśmy z założenia indukcyjnego, faktu (2.1.2) i wzoru:

$$P(A|B \cap C)P(B|C) = P(A \cap B|C).$$

□

Twierdzenie 2.1.4. *Dla łańcucha Markowa (X_n) określonego na przestrzeni stanów S, dla dowolnych $A \subseteq \mathcal{S}^n$ oraz $A_{n+1}, A_{n+2}..., A_{n+k} \subseteq S$, $k > 0$, $i_n \in S$ zachodzi:*

$$P(X_{n+k} \in A_{n+k}, X_{n+k-1} \in A_{n+k-1}...X_{n+1} \in A_{n+1} | X_n = i_n, (X_0, X_1...X_{n-1}) \in A)$$
$$= P(X_{n+k} \in A_{n+k}, X_{n+k-1} \in A_{n+k-1}...X_{n+1} \in A_{n+1} | X_n = i_n).$$

W szczególności, ponieważ A_{n+t} może być równe S, co czyni zdarzenie $X_{n+t} \in A_{n+t}$ zdarzeniem pewnym:

$$P(X_{n+k} \in A_{n+k} | X_n = i_n, (X_0, X_1...X_{n-1}) \in A) = P(X_{n+k} \in A_{n+k} | X_n = i_n).$$

Dowód. Jeśli któryś z $A_{n+1}, ...A_{n+k}$ jest pusty, to po obu stronach równości dostajemy zero, więc teza zachodzi. Jeśli żaden nie jest pusty, to z addytywności prawdopodobieństwa warunkowego wystarczy udowodnić, że dla dowolnych $i_{n+1}, ...i_{n+k} \in S$:

$$P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n, (X_0, X_1...X_{n-1}) \in A)$$
$$= P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n).$$

Niech $|A| = m > 0$ (większe od zera, gdyż zakładamy, że prawdopodobieństwo warunkowe ma sens). Dalej zapiszmy $A = \{(i_0^{(1)}, i_1^{(1)}..., i_{n-1}^{(1)}), ..., ((i_0^{(m)}, i_1^{(m)}..., i_{n-1}^{(m)})\}$. Oznaczmy kolejne elementy A przez $a_1, a_2, ...a_m$. Zauważmy, że dla każdego z tych elementów a_l:

$$P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n, (X_0, X_1...X_{n-1}) = a_l)$$
$$= P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}^{(l)}, ...X_0 = i_0^{(l)})$$
$$= P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n),$$

gdzie ostatnia równość jest na mocy lematu (2.1.3). Oznaczmy jeszcze:

$$q_l := P(X_n = i_n, (X_{n-1}, X_{n-2}...X_0) = a_l | X_n = i_n, (X_{n-1}, X_{n-2}...X_0) \in A)$$

Zauważmy, że ponieważ prawdopodobieństwo warunkowe jest miarą probabilistyczną, więc:

$$\sum_{l=1}^{m} q_l = 1.$$

Rozpisujemy na mocy twierdzenia o prawdopodobieństwie całkowitym:

$$P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n, (X_{n-1}, X_{n-2}...X_0) \in A)$$
$$= \sum_{l=1}^{m} P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n, (X_{n-1}, X_{n-2}...X_0) = a_l) \cdot q_l$$
$$= \sum_{l=1}^{m} P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n) \cdot q_l$$
$$= P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n) \cdot \sum_{l=1}^{m} q_l$$
$$= P(X_{n+k} = i_{n+k}, ...X_{n+1} = i_{n+1} | X_n = i_n),$$

co należało udowodnić. □

Wniosek 2.1.5. *Dla A, A_{n+t} oraz i_n określonych jak poprzednio, tym razem jednak dla $t = 1, 2, 3...$ (czyli nieskończonego ciągu $A_{n+1}, A_{n+2}...$) zachodzi:*

$$P(...X_{n+2} \in A_{n+2}, X_{n+1} \in A_{n+1} | X_n = i_n, (X_0, X_1...X_{n-1}) \in A) = P(...X_{n+2} \in A_{n+2}, X_{n+1} \in A_{n+1} | X_n = i_n)$$

Dowód. Mamy z ciągłości prawdopodobieństwa z góry i z (2.1.4):

$$P(...X_{n+2} \in A_{n+2}, X_{n+1} \in A_{n+1} | X_n = i_n, (X_0, X_1 ... X_{n-1}) =$$
$$\lim_{k\to\infty} P(X_{n+k} \in A_{n+k}, X_{n+k-1} \in A_{n+k-1} ... X_{n+1} \in A_{n+1} | X_n = i_n, (X_0, X_1 ... X_{n-1}) \in A) =$$
$$\lim_{k\to\infty} P(X_{n+k} \in A_{n+k}, X_{n+k-1} \in A_{n+k-1} ... X_{n+1} \in A_{n+1} | X_n = i_n) =$$
$$P(...X_{n+2} \in A_{n+2}, X_{n+1} \in A_{n+1} | X_n = i_n).$$

□

Powyższe fakty pokazują, że jeśli w ŁM znamy konkretny stan w teraźniejszości, to tylko od niego zależą stany w przyszłości (przeszłość nie ma znaczenia). Musi to być jednak konkretny stan, tj. nie można zastąpić $i_n \in S$ przez $A_n \subseteq S$, zauważmy bowiem, że np. warunek $A_n = S$ nie niesie ze sobą żadnej informacji, stąd w tej sytuacji na ogół równość nie zajdzie.

Definicja 2.1.6. *Rozkładem początkowym łańcucha Markowa* $(X_n)_{n\in\{0,1,2...\}}$

$$\boldsymbol{\nu} = (\nu_i)_{i\in S} : \nu^T \in \mathbb{R}^S$$

nazywamy rozkład prawdopodobieństwa zmiennej losowej X_0*, a więc stanu zerowego, czyli* $\nu_i = P(X_0 = i)$.

Oczywiście zachodzi:

$$\forall i \in S : \nu_i \geq 0$$

oraz:

$$\sum_{i\in S} \nu_i = 1.$$

Myślimy o $\boldsymbol{\nu}$ jako o wektorze poziomym, w którym na i-tej współrzędnej przechowujemy prawdopodobieństwo wystartowania ze stanu i.

Definicja 2.1.7. *Macierz kwadratową* $\boldsymbol{M} = (m_{ij})_{i,j\in S} \in \mathbb{R}^{S\times S}$*, taką że* $\forall i, j \in S : m_{ij} \geq 0$ *oraz* $\forall i \in S : \sum_{j\in S} m_{ij} = 1$ *nazywamy macierzą stochastyczną.*

Lemat 2.1.8. *Iloczyn macierzy stochastycznych jest macierzą stochastyczną.*

Dowód. Weźmy macierze stochastyczne $\boldsymbol{A} = (a_{ij})$ i $\boldsymbol{B} = (b_{ij})$ i oznaczmy ich iloczyn przez $\boldsymbol{AB} = ((ab)_{ij})$. $\boldsymbol{AB}$ ma wyrazy nieujemne, są one bowiem sumą iloczynów wyrazów $\boldsymbol{A}$ i $\boldsymbol{B}$. Ponadto z definicji mnożenia macierzy mamy:

$$\sum_{j\in S} (ab)_{ij} = \sum_{j,k\in S} a_{ik} b_{kj} = \sum_{k\in S} (a_{ik} \cdot \sum_{j\in S} b_{kj}) = \sum_{k\in S} (a_{ik} \cdot 1) = 1,$$

co było do udowodnienia (w ostatnich dwóch równościach skorzystaliśmy ze stochastyczności $\boldsymbol{A}$ i $\boldsymbol{B}$) □

Definicja 2.1.9. *Jednorodnym łańcuchem Markowa (**JŁM**) o rozkładzie początkowym* $(\nu_i)_{i\in S}$ *i stochastycznej macierzy (prawdopodobieństw) przejścia (w jednym kroku)* $\mathbb{P} = (p_{ij})_{i,j\in S}$ *nazywamy łańcuch Markowa* $(X_n)_{n\in\{0,1,2...\}}$*, taki że: dla* $n = 0, 1, 2...$ *oraz* $\forall i, j \in S$ *zachodzi* $P(X_{n+1} = j | X_n = i) = p_{ij}$ *(a więc wartość ta nie zależy od n).* p_{ij} *nazywamy prawdopodobieństwem przejścia (w jednym kroku) ze stanu i do stanu j.*

Twierdzenie 2.1.10. *Niech* $(X_n)_{n\in\{0,1,2...\}}$ *będzie jednorodnym łańcuchem Markowa o rozkładzie początkowym* $(\nu_i)_{i\in S}$ *oraz macierzy przejścia* $\mathbb{P} = (p_{ij})_{i,j\in S}$*. Wtedy dla każdego* $n = 1, 2, 3...$ *i* $j \in S$*:*

$$P(X_n = j) = \sum_{i_0, i_1, ... i_{n-1} \in S} \nu_{i_0} p_{i_0 i_1} p_{i_1 i_2} ... p_{i_{n-2} i_{n-1}} p_{i_{n-1} j}$$

Dowód. Przeprowadzimy dowód przez indukcję. Dla $n = 1$ z twierdzenia o prawdopodobieństwie całkowitym mamy:

$$\begin{aligned} P(X_1 = j) &= \sum_{i_0 \in S} P(X_1 = j | X_0 = i_0) P(X_0 = i_0) \\ &= \sum_{i_0 \in S} P(X_1 = j | X_0 = i_0) \cdot \nu_{i_0} \\ &= \sum_{i_0 \in S} P(X_1 = j | X_0 = i_0) \cdot \nu_{i_0} \\ &= \sum_{i_0 \in S} \nu_{i_0} \cdot p_{i_0 j}. \end{aligned}$$

Zatem twierdzenie jest prawdziwe dla $n = 1$. Dalej załóżmy, że twierdzenie jest prawdziwe dla pewnego $n \geq 1$ – pokażemy, że jest wówczas prawdziwe dla $n+1$. Korzystając kolejno z twierdzenia o prawdopodobieństwie całkowitym i z założenia:

$$\begin{aligned} P(X_{n+1} = j) &= \sum_{i_n \in S} P(X_{n+1} = j | X_n = i_n) \cdot P(X_n = i_n) \\ &= \sum_{i_n \in S} \left(p_{i_n j} \quad \cdot \sum_{i_0, i_1, \ldots i_{n-1} \in S} \nu_{i_0} p_{i_0 i_1} p_{i_1 i_2} \ldots p_{i_{n-1} i_n} \right) \\ &= \sum_{i_0, i_1, \ldots i_n \in S} \nu_{i_0} p_{i_0 i_1} p_{i_1 i_2} \ldots p_{i_{n-1} i_n} p_{i_n j}, \end{aligned}$$

kończymy dowód indukcyjny. □

Twierdzenie 2.1.11. *Oznaczmy przez $\boldsymbol{\nu^{(n)}} = (\nu_i^{(n)})_{i \in S} : \boldsymbol{\nu^{(n)}} \in \mathbb{R}^S$ rozkład prawdopodobieństwa JŁM $(X_n)_{n \in \{0,1,2\ldots\}}$ w momencie n, czyli $\nu_i^{(n)} = P(X_n = i)$. Wówczas dla każdego $n = 0, 1, 2, \ldots$ $\boldsymbol{\nu^{(n)}} = \boldsymbol{\nu} \mathbb{P}^n$.*

Dowód. Dla $n = 0$ teza wynika z definicji rozkładu początkowego, gdyż $\mathbb{P}^0$ jest równe macierzy identycznościowej. Dla $n > 0$ teza wynika z twierdzenia (2.1.10), gdyż z definicji mnożenia macierzy $\sum_{i_0, i_1, \ldots i_{n-1} \in S} \nu_{i_0} p_{i_0 i_1} p_{i_1 i_2} \ldots p_{i_{n-2} i_{n-1}} p_{i_{n-1} j}$ jest i-tym wyrazem poziomego wektora $\boldsymbol{\nu} \mathbb{P}^n$. □

Definicja 2.1.12. *Oznaczmy $\mathbb{P}(n) := \mathbb{P}^n$ dla $n = 0, 1, 2, 3\ldots$ Macierz $\mathbb{P}(n) = (p_{ij}(n))_{i,j \in S}$ będziemy nazywać macierzą (prawdopodobieństw) przejścia w n krokach, a $p_{ij}(n)$ prawdopodobieństwem przejścia z i do j w n krokach.*

Twierdzenie 2.1.13. *Zachodzą następujące własności:*
(1) Dla każdego $n = 0, 1, 2, 3\ldots$ $\mathbb{P}(n)$ jest stochastyczna
(2) Dla $m, n \in \{0, 1, 2, 3\ldots\}$ $\mathbb{P}(m+n) = \mathbb{P}(m)\mathbb{P}(n)$, czyli dla $i, j \in S$:

$$p_{ij}(m+n) = \sum_{k \in S} p_{ik}(m) \cdot p_{kj}(n)$$

(3) Dla dowolnych $m, n \in \{0, 1, 2\ldots\}$ oraz $i, j, k \in S$ zachodzi $p(m+n)_{ij} \geq p(m)_{ik} \cdot p(n)_{kj}$.
(4) Dla JŁM (X_n), $i, j \in S$, $n, m \in \{0, 1, 2\ldots\}$ zachodzi $P(X_{n+m} = j | X_n = i) = p_{ij}(m)$

Dowód. Własność (1) wynika z tego, $\mathbb{P}(n)$ jest stochastyczna dla $n = 0, 1$ ($\mathbb{P}^0 = \mathbb{P}(0) = \mathbb{I}$ jest stochastyczna) lematu (2.1.8) i prostej indukcji.
(2) wynika z łączności mnożenia macierzy (uwaga: równania z tego punktu noszą miano *równań Chapmana-Kołmogorowa*).
(3) wynika z (2) oraz tego, że prawdopodobieństwa przejść są nieujemne.
Przeprowadzimy dowód (4) i po raz kolejny będzie to dowód indukcyjny – ze względu na m: Dla $m = 0$ równość jest oczywista – zarówno lewa, jak i prawa strona jest równa 1 wtedy i tylko wtedy, gdy $i = j$ oraz 0 w przeciwnym przypadku. Załóżmy, że teza jest spełniona dla pewnego $m \geq 0$.

Wtedy dla $m+1$ z własności prawdopodobieństwa warunkowego, zał. indukcyjnego i wcześniej udowodnionych twierdzeń mamy:

$$\begin{aligned} P(X_{n+m+1}=j|X_n=i) &= \sum_{s\in S} P(X_{n+m+1}=j \wedge X_{n+m}=s|X_n=i) \\ &= \sum_{s\in S} P(X_{n+m+1}=j|X_{n+m}=s)P(X_{n+m}=s|X_n=i) \\ &= \sum_{s\in S} p_{is}(m)p_{sj} = p_{ij}(m+1), \end{aligned}$$

gdzie ostatnia równość wynika z definicji mnożenia macierzy. □

Punkt (4) powyższego twierdzenia pokazuje, że stosowane nazewnictwo jest zgodne z intuicją. $p_{ij}(n)$ w istocie zadaje prawdopodobieństwo trafienia z i do j po n krokach.

Wniosek 2.1.14. *Z powyższych rozważań oraz z twierdzenia (2.1.4): jeśli (X_n) jest JŁM o macierzy przejścia $\mathbb{P}$, to dla $t \in \mathbb{N}$ ciąg (X_{tn}) także jest JŁM – o macierzy przejścia $\mathbb{P}(t)$ i rozkładzie początkowym tym samym, co (X_n), czyli $\boldsymbol{\nu}$. Natomiast $X_k, X_{k+1}, X_{k+2}...$ jest JŁM o tej samej macierzy przejścia co (X_n), czyli $\mathbb{P}$, ale rozkładzie początkowym $\boldsymbol{\nu}^{(k)}$.*

Zdefiniujemy teraz pewne istotne własności łańcuchów Markowa, które powinny mieć łańcuchy zamodelowane do rozwiązania postawionych w tej pracy problemów.

Definicja 2.1.15. *Dla JŁM (X_n) stany $i, j \in S$ nazwiemy komunikującymi się ze sobą jeśli dla pewnych $m, n \geq 0$: $p_{ij}(m) > 0$ i $p_{ji}(n) > 0$. Oznacza to, że będąc w i z niezerowym prawdopodobieństwem trafimy kiedyś do j (i podobnie z j do i)*

Definicja 2.1.16. *Jednorodny łańcuch Markowa nazwiemy nieredukowalnym, jeśli wszystkie stany komunikują się ze sobą.*

Definicja 2.1.17. *Jednorodny łańcuch Markowa nazwiemy nieokresowym, jeśli dla każdego $i \in S$:*

$$NWD(\{n \in \{0,1,2...\} : p_{ii}(n) > 0\}) = 1$$

.

Fakt 2.1.18. *Jeśli dla $n_1, n_2, ...n_k > 0$ $p_{ii}(n_i) > 0$ dla $i = 1, 2...k$, to również $p_{ii}(a_1n_1 + a_2n_2 + ...a_kn_k) > 0$ dla nieujemnych całkowitych a_t. W szczególności jeśli $p_{ii}(n) > 0$, to również $p_{ii}(an) > 0$ dla dowolnego całkowitego $a \geq 0$.*

Faktu dowodzimy z punktu (3) twierdzenia 2.1.13 i prostej indukcji. Innymi słowy jeśli łańcuch może powrócić do tego samego stanu po pewnych czasach, to może też do niego powrócić po sumie i wielokrotności tych czasów.
Intuicyjnie można rozumieć nieokresowy łańcuch jako łańcuch, w którym stany nie mają okresu większego niż 1 – tj. nie zachodzi sytuacja, że powroty następują *tylko* po upływie całkowitych wielokrotności m dla pewnego $m > 1$. Nieredukowalność jest natomiast wyrażeniem faktu, że z każdego stanu możemy dojść do każdego innego z niezerowym prawdopodobieństwem. Warto zwrócić uwagę, że własności nieredukowalności, nieokresowości zależą tylko od macierzy przejścia i nie zależą od rozkładu początkowego, tak więc powyższe określenia będę stosował wymiennie do JŁM i jego macierzy przejścia. Z nieredukowalnością i nieokresowością wiąże się pewien lemat, który przyda się w późniejszym czasie.

Lemat 2.1.19. *Jeśli JŁM jest nieredukowalny i nieokresowy, to dla dowolnych $i, j \in S$ istnieje takie $n > 0$, że $p_{ij}(n) > 0$ i $p_{jj}(n) > 0$.*

Dowód. Z nieredukowalności łańcucha istnieje takie $m > 0$, że $p_{ij}(m) > 0$. Niech

$$A = \{n \in \{0,1,2...\} : p_{jj}(n) > 0\} = \{0, n_1, n_2, n_3...\}.$$

Wtedy z nieokresowości istnieje takie $k > 1$, że $n_1, n_2...n_k$ są względnie pierwsze. Skoro tak, to z dobrze znanego z algebry faktu [5] istnieją takie $x_1, x_2, ...x_k \in \mathbb{Z}$, że $x_1n_1 + x_2n_2 + ... + x_kn_k = 1$. Zatem będą istniały także takie całkowite nieujemne $y_1, ...y_k$, że:

$$S := y_1n_1 + y_2n_2 + ... + y_kn_k \cong -m \mod n_1$$

– wystarczy wziąć $y_l = -m \cdot x_l + t \cdot n_1$, gdzie t jest dobrane tak, żeby wszystkie y_l były nieujemne. Mamy więc $S = an_1 - m > 0$ dla pewnego $a \in \mathbb{Z}_+$. Z faktu (2.1.18) $p_{jj}(S) > 0$, a zatem z punktu (3) twierdzenia (2.1.13)

$$p_{ij}(an_1) = p_{ij}(m + S) \geq p_{ij}(m) \cdot p_{jj}(S) > 0.$$

Ale z faktu (2.1.18) również $p_{jj}(an_1) > 0$, a więc biorąc $n = an_1$ dostajemy tezę. □

Wniosek 2.1.20. *Jeśli JŁM jest nieredukowalny i nieokresowy to dla dowolnych $i, j, k \in S$ istnieje takie n, że $p_{ik}(n) > 0$ i $p_{jk}(n) > 0$.*

Dowód. Korzystamy z lematu (2.1.19) i bierzemy m, takie że $p_{ij}(m) > 0$ i $p_{jj}(m) > 0$, z nieredukowalności istnieje takie a, że $p_{jk}(a) > 0$, wystarczy zatem wziąć $n = m + a$. □

2.2 Rozkład stacjonarny i twierdzenie ergodyczne

Definicja 2.2.1. *Dla JŁM $\boldsymbol{X} = (X_n)_{n\in\{0,1,2...\}}$ rozkładem stacjonarnym nazwiemy rozkład prawdopodobieństwa na S: $\boldsymbol{\pi} = (\pi_i)_{i\in S}$ spełniający warunek:*

$$\pi_i = \sum_{j\in S} \pi_j \cdot p_{ji}$$

Co w zapisie macierzowym jest równoważne:

$$\boldsymbol{\pi} = \boldsymbol{\pi}\mathbb{P}.$$

Oczywiście jako że $\boldsymbol{\pi}$ jest rozkładem, to $\pi_i \geq 0$ oraz $\sum_{i\in S} \pi_i = 1$. Równanie z definicji powyżej nazywamy *równaniem balansu.*

Fakt 2.2.2. *Jeśli $\boldsymbol{\pi}$ jest rozkładem stacjonarnym i jednocześnie rozkładem początkowym, to rozkłady łańcucha w każdym momencie n są takie same, i.e. $\boldsymbol{\pi}^{(n)} = \boldsymbol{\pi}$ dla każdego $n = 0, 1, 2...$*

Dowód. Przez indukcję względem n. Dla $n = 0$ fakt wynika z tego, że $\boldsymbol{\pi}$ jest rozkładem początkowym. Dla $n + 1$ przy założeniu, że fakt zachodzi dla $n \geq 0$, dostajemy:

$$\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)}\mathbb{P} = \boldsymbol{\pi}\mathbb{P} = \boldsymbol{\pi}$$

□

JŁM, który w każdym kroku ma ten sam rozkład (czyli startuje z rozkładu stacjonarnego), nazywamy *stacjonarnym* – zwróćmy uwagę, że wtedy także rozkłady łączne $(X_0, X_1..., X_n)$ i $(X_h, X_{h+1}, ..., X_{h+n})$ $\forall h \in \mathbb{N}$ są takie same. Ogólnie właśnie taką własność nazywamy *stacjonarnością.*

Twierdzenie 2.2.3 (ergodyczne)**.** *Załóżmy, że JŁM (X_n) jest nieredukowalny i nieokresowy. Wówczas:*
(1) X_n ma dokładnie jeden rozkład stacjonarny $\boldsymbol{\pi}$, ponadto $\forall j \in S : \ \pi_j > 0$.
(2) Dla $i, j \in S$:

$$\lim_{n\to\infty} p_{ij}(n) = \pi_j$$

oraz dla dowolnego rozkładu początkowego $\boldsymbol{\nu}$, przyjmując wcześniejsze oznaczenia, mamy

$$\lim_{n\to\infty} \nu_j^{(n)} = \pi_j$$

(łańcuch, w którym zachodzi opisana zbieżność do rozkładu stacjonarnego, nazywamy ergodycznym).
(3) Dla stanu $j \in S$ zdefiniujmy zmienną losową

$$W_{j,n} : \Omega \to S : W_{j,n} := \frac{1}{n} \sum_{i=0}^{n-1} \mathbb{1}_{\{X_i=j\}}$$

Wówczas niezależnie od rozkładu początkowego $\forall j \in S : W_{j,n} \xrightarrow{n\to\infty} \pi_j$ prawie na pewno.

Do dowodu (1) i (2) użyjemy techniki *couplingu*. Pozwoli nam ona nie tylko na udowodnienie twierdzenia, ale także da wstępne wyobrażenie na temat tempa zbieżności algorytmów bazujących na JŁM. Do dowodu (3) będziemy musieli użyć własności tzw. *czasów n-tej wizyty w stanie s*. Da się pokazać, że zachodzenie (2) implikuje nieokresowość i nieredukowalność JŁM. Tak więc można przyjąć: ergodyczność=nieredukowalność+nieokresowość – nieredukowalną i nieokresową macierz stochastyczną będziemy zatem nazywać ergodyczną.

2.3 Konstrukcja i istnienie JŁM o zadanych rozkładzie początkowym i macierzy przejścia

W symulacjach i zastosowaniach algorytmicznych przyjmujemy, że mamy do dyspozycji generator niezależnych zmiennych losowych z rozkładu jednostajnego $U[0,1]$ (w praktyce mamy dostęp tylko do generatora liczb *pseudolosowych*, ale przy odpowiednio zaimplementowanym generatorze rozkład jednostajny jest przybliżany wystarczająco dobrze dla celów algorytmicznych).
Oznaczmy przez $U_n : n = 0, 1, 2, 3... \sim U[0,1]$ n-tą niezależną próbę z rozkładu jednostajnego (w zastosowaniach algorytmicznych wylosowaną przez generator). Dla przestrzeni stanów $S = \{s_0, s_1, ...s_k\}$, rozkładu początkowego $\boldsymbol{\nu}$ i macierzy przejścia $\mathbb{P} = (p_{ij})$. zdefiniujmy ciąg zmiennych losowych Y_n o wartościach w S:

$$Y_0 = \varphi_{\boldsymbol{\nu}}(U_0) = \begin{cases} s_0, & \text{jeśli } U_0 \in [0, \nu_{s_0}) \\ s_1, & \text{jeśli } U_0 \in [\nu_{s_0}, \nu_{s_0} + \nu_{s_1}) \\ \vdots & \\ s_l, & \text{jeśli } U_0 \in [\sum_{i=0}^{l-1} \nu_{s_i}, \sum_{i=0}^{l} \nu_{s_i}] \\ \vdots & \\ s_k, & \text{jeśli } U_0 \in [\sum_{i=0}^{k-1} \nu_{s_i}, 1] \end{cases}$$

oraz dla $n > 0$:

$$Y_n = \varphi_{\mathbb{P}}(U_n, Y_{n-1}) = \begin{cases} s_0, & \text{jeśli } U_n \in [0, p_{Y_{n-1}s_0}) \\ s_1, & \text{jeśli } U_n \in [p_{Y_{n-1}s_0}, p_{Y_{n-1}s_0} + p_{Y_{n-1}s_1}) \\ \vdots & \\ s_l, & \text{jeśli } U_n \in [\sum_{i=0}^{l-1} p_{Y_{n-1}s_i}, \sum_{i=0}^{l} p_{Y_{n-1}s_i}] \\ \vdots & \\ s_k, & \text{jeśli } U_n \in [\sum_{i=0}^{k-1} p_{Y_{n-1}s_i}, 1] \end{cases}$$

Łatwo sprawdzić, że Y_n zadaje jednorodny łańcuch Markowa o rozkładzie początkowym $\boldsymbol{\nu}$ i macierzy przejścia $\mathbb{P}$. Funkcję $\varphi_{\mathbb{P}}$ nazywamy funkcją update'u. Zauważmy, że w ten sposób otrzymujemy również dowód istnienia ciągu zmiennych losowych będącego jednorodnym łańcuchem Markowa o zadanym rozkładzie początkowym i macierzy przejścia. Podobnie, zmieniając w czasie funkcję update'u (bo zmienia się w czasie wtedy również macierz przejścia), możemy skonstruować dowolny niejednorodny łańcuch Markowa. W praktyce funkcja update'u może przyjmować różne postaci – najważniejsze, by rozkład przejść uzyskany w ten sposób był zgodny z macierzą przejścia i by opierał się na ciągu niezależnych zmiennych losowych (co daje gwarancję na spełnianie własności Markowa).

2.4 Zbieżność do rozkładu stacjonarnego. Coupling

Podrozdział zacznijmy od zdefiniowania odległości między rozkładami na S.

Definicja 2.4.1. *Odległością TV między rozkładami prawdopodobieństwa $\boldsymbol{\nu}$ i $\boldsymbol{\mu}$ na S (total variation distance) nazwiemy*

$$\|\boldsymbol{\nu} - \boldsymbol{\mu}\|_{TV} := \max_{A \subseteq S} |\boldsymbol{\nu}(A) - \boldsymbol{\mu}(A)|$$

Definicja 2.4.2. *Odległością w normie L_1 między rozkładami prawdopodobieństwa $\boldsymbol{\nu}$ i $\boldsymbol{\mu}$ na S nazwiemy:*

$$\|\boldsymbol{\nu} - \boldsymbol{\mu}\|_{L_1} = \sum_{j \in S} |\mu_j - \nu_j|$$

Jest znanym faktem, że metryka L_1 na $\mathbb{R}^S$ zadaje metrykę równoważną z metryką euklidesową – dla dowodu twierdzenia ergodycznego możemy zatem skupić się na badaniu zbieżności w L_1. Jako równoważna z euklidesową przestrzeń indukowana przez tę metrykę jest zupełna. Ponadto zbiór rozkładów tj.

$$\{\boldsymbol{\nu} \in \mathbb{R}^S : \nu_i \geq 0, \sum_{i \in S} \nu_i = 1\}$$

jest zbiorem domkniętym tej przestrzeni, a więc również jest przestrzenią zupełną w tej metryce. Poniższy lemat pokazuje równoważność metryk L_1 i TV.

Lemat 2.4.3. *Zachodzi:*

$$\|\boldsymbol{\nu} - \boldsymbol{\mu}\|_{TV} = \frac{1}{2}\|\boldsymbol{\nu} - \boldsymbol{\mu}\|_{L_1} = \sum_{\substack{j \in S: \\ \nu_j > \mu_j}} (\nu_j - \mu_j) = \sum_{\substack{j \in S: \\ \mu_j > \nu_j}} (\mu_j - \nu_j)$$

Dowód. Oznaczmy kolejno:

$$A = \{j \in S : \nu_j > \mu_j\}$$
$$B = \{j \in S : \mu_j > \nu_j\}$$
$$C = \{j \in S : \mu_j = \nu_j\}$$

Najpierw zauważmy, że dla dowolnego $D \subseteq S$

$$|\boldsymbol{\nu}(D) - \boldsymbol{\mu}(D)| = \left|\sum_{j \in D}(\nu_j - \mu_j)\right| = \left|\sum_{\substack{j \in D: \\ \nu_j > \mu_j}} (\nu_j - \mu_j) - \sum_{\substack{j \in D: \\ \mu_j > \nu_j}} (\mu_j - \nu_j)\right|$$

Ponieważ w ostatniej różnicy odjemna i odjemnik są nieujemne, więc moduł będzie największy, gdy jedno z nich będzie równe zero. Zatem zbiór, dla którego $|\boldsymbol{\nu}(D) - \boldsymbol{\mu}(D)|$ przyjmuje maksymalną wartość albo jest zawarty w $A \cup C$ albo w $B \cup C$. Ponieważ jednak stany z C nie wpływają z definicji na wartość modułu ($\nu_j - \mu_j = 0$ dla tych stanów), możemy przyjąć, że zbiór maksymalizujący wartość modułu jest w całości zawarty w A albo w całości zawarty w B. Ponadto, jeśli $D \subseteq A$, to

$$|\boldsymbol{\nu}(D) - \boldsymbol{\mu}(D)| = \sum_{j \in D}(\nu_j - \mu_j) \leq \sum_{j \in D}(\nu_j - \mu_j) + \sum_{j \in A \setminus D} (\nu_j - \mu_j) = |\boldsymbol{\nu}(A) - \boldsymbol{\mu}(A)|.$$

Analogiczne rozumowanie przeprowadzamy dla B. Stąd widać, że maksimum definiujące odległość jest przyjmowane dla A lub dla B (*).
Mamy $S = A \cup B \cup C$ oraz A, B, C są parami rozłączne. Z definicji C: $\boldsymbol{\nu}(C) = \boldsymbol{\mu}(C)$, zatem $\boldsymbol{\nu}(A) + \boldsymbol{\nu}(B) = \boldsymbol{\mu}(A) + \boldsymbol{\mu}(B)$, a więc $|\boldsymbol{\nu}(A) - \boldsymbol{\mu}(A)| = |\boldsymbol{\mu}(B) - \boldsymbol{\nu}(B)|$, czyli w połączeniu z (*):

$$\|\boldsymbol{\nu} - \boldsymbol{\mu}\|_{TV} = \sum_{\substack{j \in S: \\ \nu_j > \mu_j}} (\nu_j - \mu_j) = \sum_{\substack{j \in S: \\ \mu_j > \nu_j}} (\mu_j - \nu_j).$$

Ostatecznie z definicji normy L_1:

$$\|\boldsymbol{\nu} - \boldsymbol{\mu}\|_{L_1} = \sum_{\substack{j \in S: \\ \nu_j > \mu_j}} (\nu_j - \mu_j) + \sum_{\substack{j \in S: \\ \mu_j > \nu_j}} (\mu_j - \nu_j) = 2\|\boldsymbol{\nu} - \boldsymbol{\mu}\|_{TV},$$

co po podzieleniu stronami przez 2 kończy dowód lematu. □

Lemat 2.4.4. *Weźmy zmienne losowe X, Y o wartościach z S i o rozkładach $\boldsymbol{\nu}$ i $\boldsymbol{\mu}$ odpowiednio Niech $(X, Y) \sim \boldsymbol{\rho}$. Wówczas:*

$$P(X \neq Y) \geq \|\boldsymbol{\nu} - \boldsymbol{\mu}\|_{TV}.$$

Dowód. Jako że $\boldsymbol{\rho}$ jest rozkładem (X, Y) to $\boldsymbol{\nu}$ i $\boldsymbol{\mu}$ są jego rozkładami brzegowymi:

$$\forall i \in S : \sum_{j \in S} \rho_{i,j} = \nu_i$$
$$\forall j \in S : \sum_{i \in S} \rho_{i,j} = \mu_j.$$

Zatem $\rho_{i,i} \leq \nu_i$ oraz $\rho_{i,i} \leq \mu_i$, więc $\rho_{i,i} \leq min(\nu_i, \mu_i)$ (*). Dalej mamy

$$\begin{aligned}
P(X \neq Y) &= 1 - P(X = Y) = 1 - \sum_{i \in S} \rho_{i,i} \\
&= \sum_{i \in S} \nu_i - \sum_{i \in S} \rho_{i,i} && \text{korzystamy z (*)} \\
&\geq \sum_{i \in S} \nu_i - \sum_{\substack{i \in S \\ \nu_i \leq \mu_i}} \nu_i - \sum_{\substack{i \in S \\ \nu_i > \mu_i}} \mu_i \\
&= \sum_{\substack{i \in S \\ \nu_i > \mu_i}} (\nu_i - \mu_i) \\
&= \|\boldsymbol{\nu} - \boldsymbol{\mu}\|_{TV} && \text{z (2.4.3)}
\end{aligned}$$

□

Teraz przejdziemy do dowodu punktów (1) i (2) twierdzenia ergodycznego przy pomocy *couplingu*. Skonstruujmy dwa jednorodne łańcuchy Markowa (X_n) i (Y_n) o tej samej macierzy przejścia $\mathbb{P}$ (nieredukowalnej i nieokresowej), o rozkładach początkowych $\boldsymbol{\delta_i}$ i $\boldsymbol{\delta_j}$ odpowiednio, takich że każdy z nich jest skupiony w jednym stanie, tj. $(\boldsymbol{\delta_i})_i = 1, (\boldsymbol{\delta_i})_k = 0 \quad \forall k \neq i$ oraz $(\boldsymbol{\delta_j})_j = 1, (\boldsymbol{\delta_j})_k = 0 \quad \forall k \neq j$ (może być $i = j$ lub $i \neq j$). Używamy konstrukcji z podrozdziału (2.3). Dopóki $X_n \neq Y_n$, do generowania kolejnych stanów używamy niezależnych wzajemnie ciągów i.i.d. $U_n^{(X)}, U_n^{(Y)} \sim U[0,1]$, od pierwszego N zaś, przy którym $X_N = Y_N$, kolejne stany generujemy dla obu łańcuchów na podstawie tego samego ciągu i.i.d. zm. losowych $U_{N+1}, U_{N+2} ... \sim U[0,1]$. Dla każdej zmiennej używamy tej samej funkcji update'u $\varphi_{\mathbb{P}}$. Różne są funkcje generujące X_0 i Y_0, mianowicie $X_0 = \varphi_{\boldsymbol{\delta_i}}(U_0^{(X)}) = i$ oraz $Y_0 = \varphi_{\boldsymbol{\delta_j}}(U_0^{(Y)}) = j$.
Konstrukcja taka jest dla obu łańcuchów zgodna z założeniami konstrukcji z podrozdziału 2.3, zatem oba ciągi X_n i Y_n są łańcuchami Markowa – X_n o rozkładzie początkowym $\boldsymbol{\delta_i}$, a Y_n o $\boldsymbol{\delta_j}$. Czyli w dowolnym momencie n ich rozkłady to $\boldsymbol{\delta_i}\mathbb{P}^n$ i $\boldsymbol{\delta_j}\mathbb{P}^n$ odpowiednio. Zwróćmy teraz uwagę, że łańcuchy są skonstruowane tak, że jeśli N będzie najmniejszym takim N, że $X_N = Y_N$, to dla $n \leq N$ X_n i Y_n są niezależne, natomiast $\forall n \geq N \quad X_n = Y_n$. Dlatego też taką konstrukcję nazywamy *couplingiem*.
Ogólnie *coupling* to konstrukcja dwóch łańcuchów (dwóch 'kopii') o tej samej macierzy przejścia startujące z pewnych – zwykle różnych – rozkładów początkowych, tak że łańcuchy te po pierwszym spotkaniu biegną już razem. Stąd nazwa – od pewnego momentu bowiem łańcuchy są 'sparowane' ze sobą.

Lemat 2.4.5. *Dla powyżej zdefiniowanych łańcuchów (X_n) i (Y_n):* $\lim_{n \to \infty} P(X_n \neq Y_n) = 0$

Dowód. Jeśli $i = j$, to ciąg $q_n = P(X_n \neq Y_n)$ jest stale równy zero, więc lemat zachodzi. Załóżmy więc, że $i \neq j$. Ponieważ łańcuch jest nieredukowalny i nieokresowy, a przestrzeń stanów skończona, więc z (2.1.20) istnieją takie N, $k \in S$, że $p_{sk}(N) > 0$ dla każdego $s \in S$.

Oznaczmy $\varepsilon := \sqrt{\min\{s \in S : p_{sk}(N)\}} > 0$. Wtedy:

$$\begin{aligned}
P(X_N \neq Y_N) &\leq 1 - P(X_N = k, Y_N = k) \\
&= 1 - P(X_n = k) P(Y_n = k) && \text{z niezależności, dopóki } X_n \neq Y_n \\
&= 1 - P(X_n = k | X_0 = i) \cdot P(Y_n = k | Y_0 = j) \\
&= 1 - p_{ik} p_{jk} \leq 1 - \sqrt{\varepsilon^2} = 1 - \varepsilon < 1
\end{aligned}$$

Pokażemy teraz, że $P(X_{tN} \neq Y_{tN}) \leq (1-\varepsilon)^t$ dla $t = 1, 2, 3...$ (*). Przeprowadzimy argument indukcyjny. Dla $t = 1$, jak pokazaliśmy, jest to prawda. Przy założeniu, że twierdzenie jest prawdziwe dla wszystkich t: $T \geq t \geq 0$, dla $T+1$ mamy

$$\begin{aligned} P(X_{(T+1)N} \neq Y_{(T+1)N}) &= P(X_{(T+1)N} \neq Y_{(T+1)N} \wedge X_{TN} \neq Y_{TN}) \qquad \text{z konstrukcji łańcuchów} \\ &= P(X_{(T+1)N} \neq Y_{(T+1)N} | X_{TN} \neq Y_{TN}) P(X_{TN} \neq Y_{TN}) \\ &\leq (1 - (P(X_{(T+1)N} = k, P(Y_{(T+1)N)} = k | X_{TN} \neq Y_{TN})) \cdot P(X_{TN} \neq Y_{TN}) \end{aligned}$$

z założenia indukcyjnego i własności prawdopodobieństwa warunkowego:

$$= \left(1 - \sum_{\substack{i,j \in S \\ i \neq j}} P(X_{(T+1)N} = k, P(Y_{(T+1)N} = k | X_{TN} = i, Y_{TN} = j) P(X_{TN} = i, Y_{TN} = j | X_{TN} \neq Y_{TN}) \right)$$

$$\cdot (1-\varepsilon)^T$$

z definicji ε, niezależności, dopóki $X_n \neq Y_n$ i jednorodności (2.1.14):

$$\leq \left(1 - \varepsilon \left(\sum_{\substack{i,j \in S \\ i \neq j}} P(X_{NT} = i, Y_{NT} = j | X_{NT} \neq Y_{NT}) \right) \right) \cdot (1-\varepsilon)^T$$

w sumie dostajemy całe zdarzenie, po którym warunkujemy, więc:

$$= (1-\varepsilon)(1-\varepsilon)^T = (1-\varepsilon)^{T+1}.$$

Teraz zauważmy jeszcze, że ciąg $q_n = P(X_n \neq Y_n)$ jest nierosnący, ponieważ $\{X_{n+1} \neq Y_{n+1}) \subseteq \{X_n \neq Y_n)$, ze względu na to jak zdefiniowane są oba łańcuchy Markowa. Stąd dla dowolnego $M > 0$ od pewnego miejsca wszystkie $q_n \leq (1-\varepsilon)^M$, a więc $q_n \xrightarrow{n \to \infty} 0$. □

Twierdzenie 2.4.6. *Dla niereducowalnej i nieokresowej macierzy $\mathbb{P}$ oraz dowolnych $i, j \in S$ zachodzi:*

$$\|\boldsymbol{\delta_i}\mathbb{P}^n - \boldsymbol{\delta_j}\mathbb{P}^n\|_{TV} \xrightarrow{n \to \infty} 0$$
$$\|\boldsymbol{\delta_i}\mathbb{P}^n - \boldsymbol{\delta_j}\mathbb{P}^n\|_{L_1} \xrightarrow{n \to \infty} 0.$$

Dowód. Korzystamy z (2.4.4) i dostajemy w ten sposób, że

$$0 \leq \|\boldsymbol{\delta_i}\mathbb{P}^n - \boldsymbol{\delta_j}\mathbb{P}^n\|_{TV} \leq q_n = P(X_n \neq Y_n)$$

z (2.4.5) $q_n \to 0$, więc korzystamy z twierdzenia o trzech ciągach i dostajemy pierwszą część lematu. Do części drugiej wystarczy przypomnieć sobie, że z (2.4.3):

$$\|\boldsymbol{\delta_i}\mathbb{P}^n - \boldsymbol{\delta_j}\mathbb{P}^n\|_{L_1} = 2\|\boldsymbol{\delta_i}\mathbb{P}^n - \boldsymbol{\delta_j}\mathbb{P}^n\|_{TV} \to 0$$

□

Widać więc kandydata na rozkład stacjonarny

$$\boldsymbol{\pi} := \lim_{n \to \infty} \boldsymbol{\delta_i}\mathbb{P}^n,$$

gdzie wybór $i \in S$ jest dowolny, z (2.4.5) bowiem można zauważyć, że jeśli granica istnieje, to nie zależy od i. Żeby udowodnić punkty (1), (2) twierdzenia ergodycznego (2.2.3), pozostaje udowodnić poniższe twierdzenie:

Twierdzenie 2.4.7. *Zachodzą następujące własności:*

(a) rozważana granica istnieje i nie zależy od rozkładu początkowego

(b) zadaje ona rozkład stacjonarny ($\boldsymbol{\pi}$)

(c) zachodzi $p_{ij}(n) \to \pi_j$

(d) $\pi_i > 0$ dla każdego $i \in S$

Dowód. (a): Pokażemy, że ciąg $\boldsymbol{\delta_i}\mathbb{P}^n$ spełnia warunek Cauchy'ego. Istotne jest, że wcześniejsze rozważania były niezależne od stanów początkowych i, j, zatem (2.4.5) zachodzi dla dowolnej pary stanów. Ustalmy $\varepsilon > 0$ oraz takie K, że dla każdej pary stanów $s_1, s_2 \in S$ i $n > K$ zachodzi $\|\boldsymbol{\delta_{s_1}}\mathbb{P}^n - \boldsymbol{\delta_{s_2}}\mathbb{P}^n\|_{L1} < \varepsilon$. Takie K istnieje, bo jest skończenie wiele par stanów i dla każdej pary $\|\boldsymbol{\delta_{s_1}}\mathbb{P}^n - \boldsymbol{\delta_{s_2}}\mathbb{P}^n\|_{L1} \to 0$ z (2.4.6). Weźmy sobie dowolne $N > M > K$. Zauważmy tedy, że $\boldsymbol{\delta_i}\mathbb{P}^{N-M}$ zadaje pewien rozkład $\boldsymbol{\mu}$. Zauważmy też, że taki rozkład $\boldsymbol{\mu}$ (i dowolny inny) na S da się zapisać jako $\boldsymbol{\mu} = \sum_{k\in S} \mu_k \boldsymbol{\delta_k}$. Zatem:

$$\begin{aligned}
\|\boldsymbol{\delta_i}\mathbb{P}^N - \boldsymbol{\delta_i}\mathbb{P}^M\|_{L_1} &= \|(\boldsymbol{\delta_i}\mathbb{P}^{N-M})\mathbb{P}^M - \boldsymbol{\delta_i}\mathbb{P}^M\|_{L_1} \\
&= \|(\boldsymbol{\mu}\mathbb{P}^M - \boldsymbol{\delta_i}\mathbb{P}^M\|_{L_1} \\
&= \|(\boldsymbol{\mu}\mathbb{P}^M - \boldsymbol{\delta_i}\mathbb{P}^M\|_{L_1} \\
&= \|(\sum_{k\in S} \mu_k \boldsymbol{\delta_k})\mathbb{P}^M - \boldsymbol{\delta_i}\mathbb{P}^M\|_{L_1} \\
&= \|(\sum_{k\in S} \mu_k \boldsymbol{\delta_k})\mathbb{P}^M - (\sum_{k\in S} \mu_k)\boldsymbol{\delta_i}\mathbb{P}^M\|_{L_1} \\
&= \|(\sum_{k\in S} \mu_k (\boldsymbol{\delta_k} - \boldsymbol{\delta_i}))\mathbb{P}^M\|_{L_1} \\
&\leq \sum_{k\in S} \mu_k \|\boldsymbol{\delta_k}\mathbb{P}^M - \boldsymbol{\delta_i}\mathbb{P}^M\|_{L1} < \sum_{k\in S} \mu_k \varepsilon = \varepsilon,
\end{aligned}$$

gdzie przedostatnia nierówność wynika z nierówności trójkąta, a ostatnia z tego, że $M > K$. Zatem warunek Cauchy'ego jest spełniony, przestrzeń rozkładów jest zupełna, a więc granica istnieje. Przeprowadzając rozumowanie jak powyżej dla dowolnego rozkładu początkowego:

$$\boldsymbol{\nu} = \sum_{k\in S} \nu_k \boldsymbol{\delta_k}$$

(czyli wstawiając $\boldsymbol{\nu}$ w miejsce $\boldsymbol{\mu}$), pokazujemy, że

$$\boldsymbol{\nu}^{(n)} \xrightarrow{n\to\infty} \boldsymbol{\pi}.$$

Zatem granica nie zależy od rozkładu początkowego. □

Dowód. (b): Zauważmy, że

$$\boldsymbol{\pi}\mathbb{P} = (\lim_{n\to\infty} \boldsymbol{\delta_i}\mathbb{P}^n)\mathbb{P} = \lim_{n\to\infty} \boldsymbol{\delta_i}\mathbb{P}^{n+1} = \boldsymbol{\pi},$$

zatem π jest rozkładem stacjonarnym. Jest też jedynym rozkładem stacjonarnym, bo pokazaliśmy wcześniej, że granica

$$\lim_{n\to\infty} \boldsymbol{\nu}\mathbb{P}^n = \boldsymbol{\pi}$$

nie zależy od rozkładu początkowego. Jeśli istniałby inny rozkład stacjonarny $\boldsymbol{\pi'} \neq \boldsymbol{\pi}$, to z równania balansu byłoby $\boldsymbol{\pi'}\mathbb{P} = \boldsymbol{\pi'}$, a więc $\boldsymbol{\pi'}\mathbb{P}^n \to \boldsymbol{\pi'}$, ale skoro granica nie zależy od rozkładu początkowego, to $\boldsymbol{\pi'}\mathbb{P}^n \to \boldsymbol{\pi}$, a więc mamy sprzeczność. Zatem rozkład $\boldsymbol{\pi}$ jest jedynym rozkładem stacjonarnym. □

Dowód. (c): Ustalmy $i, j \in S$. Mamy

$$\pi_j = (\lim_{n\to\infty} \boldsymbol{\delta_i}\mathbb{P}^n)_j = (\boldsymbol{\delta_i} \lim_{n\to\infty} \mathbb{P}^n)_j = (\boldsymbol{\delta_i})_i \lim_{n\to\infty} p_{ij}(n) = \lim_{n\to\infty} p_{ij}(n)$$

co należało udowodnić. □

Dowód. (d): Weźmy dowolne $i \in S$. Jak w lemacie (2.4.5): istnieje N, że $\forall s \in S : p_{si}(N) > 0$, ustalmy $\varepsilon = \min_{s\in S}(p_{si}(N)) > 0$. Z definicji ε i własności Markowa

$$\begin{aligned} P(X_N = i) &= \sum_{s\in S} P(X_N = i|X_0 = s)P(X_0 = s) \\ &= \sum_{s\in S} p_{si}(N)P(X_0 = s) \\ &\geq \varepsilon \sum_{s\in S} P(X_0 = s) = \varepsilon. \end{aligned}$$

Stosując prostą indukcję pokazujemy, że także dla każdego $t = 1, 2, 3...$, $P(X_{tN} = i) \geq \varepsilon$, bo

$$P(X_{(t+1)N} = i) = \sum_{s\in S} p_{si}(N)P(X_{tN} = s) \geq \varepsilon \sum_{s\in S} P(X_{tN} = s) = \varepsilon.$$

Zatem $\lim_{n\to\infty} \nu_i^{(n)} \geq \varepsilon > 0$ i stąd $\pi_i > 0$. □

Udowodnione zostały pierwsze dwa punkty twierdzenia ergodycznego. Widzimy również, że dowód lematu (2.4.5), przedstawienie dowolnego rozkładu jako sumy ważonej $\boldsymbol{\delta_k}$ oraz nierówność z (2.4.4) wskazują w pewien sposób tempo zbieżności do rozkładu stacjonarnego, tj. odległość od rozkładu stacjonarnego w n-tym kroku jest mniejsza od $c \cdot (\sqrt[N]{(1-\varepsilon)})^n = c\alpha^n$ dla N, ε zdefiniowanych w dowodzie (2.4.5) i pewnej stałej c. Tak więc tempo zbieżności wg rozkładu jest w pewien sposób wykładnicze, jednak na razie nie mówi to nam za dużo. Nieco dokładniej tempem zbieżności zajmę się przy okazji analizy algorytmów (pojawią się pojęcia takie jak np. *mixing time*).

2.5 Czasy n-tej wizyty i zbieżność prawie na pewno

Przyjmujemy te same założenia na temat JŁM, co w poprzednim rozdziale (czyli nieredukowalność i nieokresowość, skończona przestrzeń stanów S).

Definicja 2.5.1. *Czasem n-tej wizyty: $\tau_n^{(s)}$ w stanie $s \in S$ jednorodnego łańcucha Markowa X_n. nazywamy:*

$$\begin{aligned} \tau_1^{(s)} &= \inf_{m=0,1,2...} \{m : X_m = s\} \\ \tau_n^{(s)} &= \inf_{m>\tau_{n-1}^{(s)}} \{m : X_m = s\} \ \ dla\ n > 1 \end{aligned}$$

Ponadto wprowadźmy:

$$\begin{aligned} \gamma_1^{(s)} &= \tau_1^{(s)} \\ \gamma_n^{(s)} &= \tau_n^{(s)} - \tau_{n-1}^{(s)} \ \ dla\ n > 1 \end{aligned}$$

Intuicja jest jasna: $\tau_n^{(s)}$ to moment, w którym łańcuch odwiedził stan s po raz n-ty, $\gamma_1^{(s)}$ to czas, jaki upłynął do pierwszej wizyty łańcucha w stanie s, $\gamma_2^{(s)}$ to czas, który upłynął między pierwszą a drugą wizytą, kolejne γ to czasy między kolejnymi powrotami.

By móc wykorzystać te struktury, zaczniemy od udowodnienia, że można je traktować jak zmienne losowe, czyli że są skończone prawie na pewno. Zaczniemy od lematu.

Lemat 2.5.2. *Istnieją $1 \geq \varepsilon > 0$ i $N \in \mathbb{N}$ takie że $\forall s \in S$:*

$$Q_t := P(X_{tN} \neq s, X_{(t-1)N} \neq s...X_N \neq s) \leq (1-\varepsilon)^t.$$

Dowód. W zasadzie dowód powyższego lematu zawiera się w dowodzie lematu (2.4.4). Jak poprzednio, istnieje takie $N > 0$, że dla dowolnego $j \in S$ mamy $p_{js}(N) > 0$, a więc bierzemy

$1 \geq \varepsilon = \min_{j \in S}\{p_{js}(N)\} > 0$ i przeprowadzamy dowód indukcyjny.
dla t = 1 mamy ($\boldsymbol{\nu}$ – rozkład początkowy):

$$Q_1 = 1 - P(X_N = s) = 1 - \sum_{j \in S} \nu_j p_{js}(N) \leq 1 - \varepsilon \sum_{j \in S} \nu_j = 1 - \varepsilon$$

Dla $t > 1$ przy założeniu, że lemat zachodzi dla $t - 1$:

$$\begin{aligned} Q_t &= P(X_{tN} \neq s, X_{(t-1)N} \neq s, ...X_N \neq s) \\ &= P(X_{tN} \neq s | X_{(t-1)N} \neq s, ...X_N \neq s) \cdot P(X_{(t-1)N} \neq s, ...X_N \neq s) \\ &= (1 - P(X_{tN} = s | X_{(t-1)N} \neq s, ...X_N \neq s)) \cdot P(X_{(t-1)N} \neq s, ...X_N \neq s) \\ &\leq (1 - \varepsilon) Q_{t-1} \\ &\leq (1 - \varepsilon)(1 - \varepsilon)^{t-1} = (1 - \varepsilon)^t, \end{aligned}$$

gdzie pierwszą nierówność dostajemy łatwo z rozbicia zbioru, po którym warunkujemy, własności Markowa i ograniczenia prawdopodobieństwa przejścia przez ε z dołu, a druga nierówność to oczywiście założenie indukcyjne. □

Wniosek 2.5.3. *Dla każdego $s \in S$: $P(X_1 \neq s, X_2 \neq s...) = 0$.*

Dowód. Weźmy takie N i ε jak w lemacie (2.5.2). Wtedy z zawierania zdarzeń dla $t = 1, 2, 3, ...$:

$$0 \leq P(X_1 \neq s, X_2 \neq s...) \leq P(X_N \neq s, X_{2N} \neq s, ..., X_{tN} \neq s) = Q_t \leq (1 - \varepsilon)^t.$$

t jest dowolnie duże, a prawa strona dąży do zera przy t dążącym do nieskończoności, bo

$$0 \leq 1 - \varepsilon < 1,$$

więc dostajemy tezę. □

Twierdzenie 2.5.4. *Dla każdego $s \in S$: $\tau_t^{(s)}$, $t = 1, 2, 3, ...$ są skończone prawie na pewno.*

Dowód. Przeprowadzimy dowód indukcyjny ze względu na t. Z (2.5.2) mamy:

$$\begin{aligned} 1 \geq P(\tau_1^{(s)} < \infty) &= P(\exists n \in \{0, 1, 2, 3...\} : X_n = s) \\ &= 1 - P(X_0 \neq s, X_1 \neq s, X_2 \neq s...) \\ &\geq 1 - P(X_1 \neq s, X_2 \neq s, X_3 \neq s...) \\ &= 1 \qquad \text{z wniosku (2.5.3)} \end{aligned}$$

Zatem dla $t = 1$ twierdzenie zachodzi. Zauważmy prosty fakt (*):

$$\{\tau_t^{(s)} = n\} = \{X_n = s \text{ oraz } X_k = s \text{ dla dokładnie } t - 1 \text{ spośród } k = 0, 1, 2, ...n - 1\}$$

Teraz przy założeniu prawdziwości twierdzenia dla $\tau_t^{(s)}$, dla $\tau_{t+1}^{(s)}$ dostajemy:

$$\begin{aligned} 1 \geq P(\tau_{t+1}^{(s)} < \infty) &= \sum_{n=0}^{\infty} P(\tau_{t+1}^{(s)} < \infty | \tau_t^{(s)} = n) P(\tau_t^{(s)} = n) \\ &= \sum_{n : P(\tau_t^{(s)} = n) > 0} (1 - P(...X_{n+2} \neq s, X_{n+1} \neq s | \tau_t^{(s)} = n)) P(\tau_t^{(s)} = n) \end{aligned}$$

z własności Markowa, (*) i wniosku (2.1.5):

$$= \sum_{n : P(\tau_t^{(s)} = n) > 0} (1 - P(...X_{n+2} \neq s, X_{n+1} \neq s | X_n = s)) P(\tau_t^{(s)} = n)$$

z wniosku (2.1.14) oraz zastosowania (2.5.3) dla JŁM $(X_n, X_{n+1}, ...)$ o rozkładzie początkowym – czyli rozkładzie X_n – skupionym w s:

$$= 1 \quad \cdot \sum_{n: P(\tau_t^{(s)} = n) > 0} P(\tau_t^{(s)} = n)$$

z zał. indukcyjnego:

$$= \sum_{n: P(\tau_t^{(s)} = n) > 0} P(\tau_t^{(s)} = n) \quad = \quad 1.$$

Stąd:

$$P(\tau_{t+1}^{(s)} < \infty) = 1,$$

co należało udowodnić. W szczególności zauważmy, że z twierdzenia w prosty sposób wynika, że γ również są skończone prawie na pewno. □

Następnie pokażemy, że dla każdego momentu n-tej wizyty istnieje wartość oczekiwana. Przydać się może klasyczny lemat na temat wartości oczekiwanej.

Lemat 2.5.5. *Niech Z będzie zmienną losową przyjmującą tylko wartości ze zbioru $0, 1, 2....$ Wówczas:*

$$\mathbb{E}|Z| = \mathbb{E}Z = \sum_{n=0}^{\infty} P(Z > n)$$

Dowód. Pierwsza równość jest oczywista, bo zmienna przyjmuje tylko wartości nieujemne. Dalej:

$$\begin{aligned}
\mathbb{E}Z &= 0 \cdot P(Z = 0) + 1 \cdot P(Z = 1) + 2 \cdot P(Z = 2) + 3 \cdot P(Z = 3)... \\
&= (P(Z = 1) + P(Z = 2) + ...) + (P(Z = 2) + P(Z = 3) + ...) + (P(Z = 3) + ...) + ... \\
&= P(Z > 0) + P(Z > 1) + ... = \sum_{n=0}^{\infty} P(Z > n).
\end{aligned}$$

Można było zmieniać kolejność sumowania, wszystkie wyrazy bowiem są nieujemne, stąd nie ma to wpływu na wartość nieskończonej sumy. □

Twierdzenie 2.5.6. *Dla każdego $s \in S$, dla $\gamma_1^{(s)}, \gamma_2^{(s)}...$ istnieje wartość oczekiwana.*

Dowód. Zwróćmy uwagę, że wszystkie γ przyjmują tylko wartości ze zbioru $0, 1, 2...$, można więc użyć lematu (2.5.5). Zacznijmy od $\gamma_1^{(s)}$, korzystając z N, ε opisanych w (2.5.2).

$$\begin{aligned}
\mathbb{E}(|\gamma_1^{(s)}|) = \mathbb{E}(\gamma_1^{(s)}) &= \mathbb{E}(\tau_1^{(s)}) \\
&= \sum_{l=0}^{\infty} P(\tau_1^{(s)} > l) \\
&= \sum_{l=0}^{\infty} P(X_0 \neq s, ..., X_l \neq s) \\
&= \sum_{l=0}^{N-1} P(X_0 \neq s, ..., X_l \neq s) + \sum_{l=N}^{2N-1} P(X_0 \neq s, ..., X_l \neq s)... \\
&\leq \sum_{l=0}^{N-1} 1 + \sum_{l=N}^{2N-1} P(X_N \neq s) + \sum_{l=2N}^{3N-1} P(X_{2N} \neq s, X_N \neq s) + ... \\
&\text{z lematu (2.5.2)} \\
&\leq N(1 + (1 - \varepsilon) + (1 - \varepsilon)^2...) = N/\varepsilon < \infty.
\end{aligned}$$

Weźmy teraz $n > 1$ i obliczmy $\mathbb{E}(\gamma_n^{(s)} | \tau_{n-1}^{(s)} = t)$ (nie wprowadzam w tej pracy pojęcia warunkowej wartości oczekiwanej czy warunkowego prawdopodobieństwa jako zmiennych losowych, a korzystam z ich klasycznych definicji, tak że przyjmujemy $P(\tau_{n-1}^{(s)} = t) > 0$):

$$\mathbb{E}(\gamma_n^{(s)} | \tau_{n-1}^{(s)} = t) = \mathbb{E}(\tau_n^{(s)} - \tau_{n-1}^{(s)} | \tau_{n-1}^{(s)} = t) =$$

oczywiście z definicji mamy $\tau_n^{(s)} - \tau_{n-1}^{(s)} > 0$, więc:

$$= \sum_{l=1}^{\infty} l \cdot P(\tau_n^{(s)} - \tau_{n-1}^{(s)} = l | \tau_{n-1}^{(s)} = t)$$

z definicji τ i z twierdzenia (2.1.4):

$$= \sum_{l=1}^{\infty} l \cdot P(X_{t+l} = s, ..., X_{t+2} \neq s, X_{t+1} \neq s | X_t = s) =$$

ponownie korzystamy z wcześniej zdefiniowanych N i ε:

$$\sum_{l=1}^{N} l \cdot P(X_{t+l} = s, ..., X_{t+2} \neq s, X_{t+1} \neq s | X_t = s) + \sum_{l=N+1}^{2N} l \cdot P(X_{t+l} = s, ..., X_{t+2} \neq s, X_{t+1} \neq s | X_t = s) ... \leq$$
$$N \left(\sum_{l=1}^{N} 1 \right) + 2N \left(\sum_{l=N+1}^{2N} P(X_{t+N} \neq s | X_t = s) \right) + 3N \left(\sum_{l=2N+1}^{3N} P(X_{t+2N} \neq s, X_{t+N} \neq s | X_t = s) \right) ... \leq$$

z lematu (2.5.2) zastosowanego dla JŁM $X_t, X_{t+1}...$ z rozkładem początkowym skupionym w s:

$$\leq N^2 + 2N^2(1-\varepsilon) + 3N^2(1-\varepsilon)^2 ... = N^2(1 + 2(1-\varepsilon) + ...) ... =$$
$$= \frac{N^2}{(1-\varepsilon)^2} < \infty.$$

Końcowa równość to znany wzór na sumę szeregu np^{n-1} dla $|p| < 1$. Widać, że dla wszystkich warunków $\tau_{n-1}^{(s)} = t$ warunkowe wartości oczekiwane γ są wspólnie ograniczone, a ponieważ wg wcześniejszego twierdzenia wszystkie takie warunki dają w sumie prawdopodobieństwo jeden, to także cała wartość oczekiwana jest ograniczona przez tę samą wartość. Zatem istnieje skończona wartość oczekiwana dla wszystkich $\gamma_n^{(s)}$, co należało udowodnić. Łatwo zauważyć, że stąd również wynika istnienie skończonej wartości oczekiwanej dla wszystkich τ. □

Przeprowadzając podobne rozumowanie, można udowodnić, że także dowolny moment zmiennych $\gamma_1^{(s)}, \gamma_2^{(s)}, ...$ istnieje. Istotnie:

Fakt 2.5.7. *Dla $k = 1, 2, 3...$, $n = 1, 2, 3...$: $\mathbb{E}(|\gamma_n^{(s)}|^k) < \infty$.*

Dowód. Skorzystamy znowu z ε i N. Dla $\gamma_1^{(s)}$:

$$\mathbb{E}\left(\left| \gamma_1^{(s)} \right|^k \right) = \mathbb{E}\left(\left(\gamma_1^{(s)} \right)^k \right) = \mathbb{E}\left(\left(\tau_1^{(s)} \right)^k \right)$$
$$\leq \sum_{l=0}^{\infty} l^k P(\tau_1^{(s)} = l)$$
$$= \sum_{l=0}^{\infty} l^k P(X_l = s, X_{l-1} \neq s ..., X_1 \neq s, X_0 \neq s)$$
$$\leq N^k \left(\sum_{l=1}^{N} 1 \right) + (2N)^k \left(\sum_{l=N+1}^{2N} P(X_N \neq s) \right) + (3N)^k \left(\sum_{l=2N+1}^{3N} P(X_N \neq s, X_{2N} \neq s) \right) +$$
$$...$$
$$\text{z lematu (2.5.2)}$$
$$\leq N^{k+1}(1^k + 2^k(1-\varepsilon) + 3^k(1-\varepsilon)^2 ...) < \infty,$$

gdzie ostatnia suma jest skończona na mocy kryterium Cauchy'ego zbieżności szeregów, bo

$$\sqrt[n]{(n+1)^k(1-\varepsilon)^n} \xrightarrow{n\to\infty} (1-\varepsilon) < 1$$

Podobnie postępujemy dla $n = 2, 3, 4....$ Powtarzając kroki z poprzedniego dowodu: dla każdego warunku $\tau_{n-1}^{(s)} = t$, takiego że $P(\tau_{n-1}^{(s)} = t) > 0$, dostajemy:

$$\mathbb{E}\left(\left(\gamma_n^{(s)}\right)^k |\tau_{n-1}^{(s)} = t\right) \leq N^k \left(\sum_{l=1}^{N} 1\right) + (2N)^k \left(\sum_{l=N+1}^{2N} P(X_{t+N} \neq s | X_t = s)\right) +$$
$$(3N)^k \left(\sum_{l=2N+1}^{3N} P(X_{t+2N} \neq s, X_{t+N} \neq s | X_t = s)\right) ... \leq$$
$$N^{k+1} + 2^k N^{k+1}(1-\varepsilon) + 3^k N^{k+1}(1-\varepsilon)^2 ... = N^{k+1}(1 + 2^k(1-\varepsilon) + 3^k(1-\varepsilon)^2 ...) < \infty,$$

gdzie stosujemy (2.5.2) dla odpowiedniego łańcucha, a skończoność sumy znowu wynika z kryterium Cauchy'ego. Korzystając ze skończoności prawie na pewno τ i sumując po wszystkich warunkach, dostajemy tezę. Łatwo zauważyć, że w takim razie istnieją skończone momenty dla wszystkich $\tau_1^{(s)}, \tau_2^{(s)} ...$ □

Teraz formalnie pokażemy intuicyjny fakt, że czasy, które upływają między kolejnymi wizytami w s są niezależne i mają ten sam rozkład.

Twierdzenie 2.5.8. *$\gamma_2^{(s)}, \gamma_3^{(s)} ...$ mają ten sam rozkład.*

Dowód. Niech $n \geq 2$, $k = 1, 2, 3...$, rozbijamy prawdopodobieństwo względem $\tau_{n-1}^{(s)}$ podobnie jak w dowodzie poprzedniego twierdzenia (korzystając z twierdzenia (2.1.4)):

$$P(\gamma_n^{(s)} = k) = \sum_{P(\tau_{n-1}^{(s)} = t) > 0} P(X_{t+k} = s, ..., X_{t+2} \neq s, X_{t+1} \neq s | X_t = s) P(\tau_{k-1}^{(s)} = t) =$$

z jednorodności łańcucha dla każdego t, ze skończoności τ p.n.:

$$= P(X_k = s, ..., X_2 \neq s, X_1 \neq s | X_0 = s) \left(\sum_{P(\tau_{n-1}^{(s)} = t) > 0} P(\tau_{k-1}^{(s)} = t)\right) =$$

$$\sum_{\substack{i_1, i_2 ... i_{k-1} \in S \\ i_1, ..., i_{k-1} \neq s}} P(X_k = s, X_{k-1} = i_{k-1}, ..., X_1 = i_1 | X_0 = s) =$$
$$\sum_{i_1, i_2 ... i_{k-1} \in S \setminus \{s\}} p_{s i_1} p_{i_1 i_2} ... p_{i_{k-2} i_{k-1}} p_{i_{k-1} s} = C(\mathbb{P}, k, s),$$

gdzie C jest pewną funkcją macierzy $\mathbb{P}$, liczby kroków k i stanu s. Widać zatem, że rozkład nie zależy w żaden sposób od n dla $n = 2, 3, ...$, co należało udowodnić. W szczególności $\gamma_2^{(s)}, \gamma_3^{(s)}, ...$ mają tę samą wartość oczekiwaną. □

Dalej dla $n \geq 2$ przyjmujemy takie właśnie oznaczenie: $C(\mathbb{P}, k, s) := P(\gamma_n^{(s)} = k)$.

Twierdzenie 2.5.9. *$\gamma_2^{(s)}, \gamma_3^{(s)} ...$ to ciąg niezależnych zmiennych losowych.*

Dowód. Zauważmy, że dla każdego $k \geq 2$ i $0 < t_2, ..., t_k \in \mathbb{N}$:

$$P(\gamma_k^{(s)} = t_k, \gamma_{k-1}^{(s)} = t_{k-1} ..., \gamma_2^{(s)} = t_2) =$$

$$\sum_{t : P(\tau_1^{(s)} = t) > 0} P(\gamma_k^{(s)} = t_k, \gamma_{k-1}^{(s)} = t_{k-1} ..., \gamma_2^{(s)} = t_2 | \tau_1^{(s)} = t)) P(\tau_1^{(s)} = t) =$$

$$\sum_{t:P(\tau_1^{(s)}=t)>0} P(X_{t+t_2+...t_k}=s, X_{t+t_2+...t_k-1}\neq s...X_{t+t_2+...t_{k-1}+1}\neq s...X_{t+t_2+...t_{k-1}}=s,...$$
$$X_{t+t_2}=s, X_{t+t_2-1}\neq s...X_{t+1}\neq s|X_t=s)P(\tau_1^{(s)}=t)=$$

$$\sum_{t:P(\tau_1^{(s)}=t)} C(\mathbb{P},t_2,s)C(\mathbb{P},t_3,s)...C(\mathbb{P},t_k,s)P(\tau_1^{(s)}=t)=$$
$$C(\mathbb{P},t_2,s)C(\mathbb{P},t_3,s)...C(\mathbb{P},t_k,s)=P(\gamma_k^{(s)}=t_k)P(\gamma_{k-1}^{(s)}=t_{k-1})...P(\gamma_2^{(s)}=t_2)$$

W przedostatniej równości skorzystaliśmy z (2.5.8), tożsamości:

$$P(A_n,...A_1|A_0)=P(A_n|A_{n-1},...A_0)P(A_{n-1}|A_{n-2}...A_0)...P(A_1|A_0)$$

oraz własności Markowa i twierdzenia (2.1.4). Udowodnioną własność możemy łatwo uogólnić z $t_2,...t_k\in\mathbb{N}$ na $A_2,...A_k\subseteq\mathbb{N}$, rozbijając każdy A_n na pojedyncze elementy i sumując, zatem mamy:

$$P(\gamma_k^{(s)}\in A_k,\gamma_{k-1}^{(s)}\in A_{k-1}...,\gamma_2^{(s)}\in A_2)=P(\gamma_k^{(s)}\in A_k)P(\gamma_{k-1}^{(s)}\in A_{k-1})...P(\gamma_2^{(s)}\in A_2),$$

co dla **ciągu** zmiennych losowych jest równoważne niezależności. □

Zbliżamy się do końca dowodu (3) z twierdzenia ergodycznego. Potrzebujemy jeszcze wersji mocnego prawa wielkich liczb. Podaję ją bez dowodu, dowód można znaleźć np. w [6].

Twierdzenie 2.5.10 (Mocne prawo wielkich liczb Kołmogorowa, MPWL)**.** *Dany jest ciąg i.i.d. zmiennych losowych $X_1,X_2,...$, taki że $\mathbb{E}(X_1)$ istnieje i jest skończona, wtedy:*

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}X_i=\mathbb{E}(X_1)$$

prawie na pewno.

Oznaczmy teraz przez $\lambda_j:=\mathbb{E}\gamma_2^{(j)}$.

Lemat 2.5.11.

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}\gamma_i^{(j)}=\lambda_j$$

prawie na pewno.

Dowód.

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}\gamma_i^{(j)}=\lim_{n\to\infty}\left(\frac{\gamma_1^{(j)}}{n}+\frac{n-1}{n}\cdot\frac{1}{n-1}\sum_{i=2}^{n}\gamma_i^{(j)}\right)$$
$$=\lim_{n\to\infty}\frac{1}{n-1}\sum_{i=2}^{n}\gamma_i^{(j)}=\lambda_j,$$

gdzie skorzystaliśmy z tego, że $\gamma_1^{(j)}$ jest skończona prawie na pewno i MPWL dla $\gamma_2^{(j)},\gamma_3^{(j)}...$, które, jak pokazaliśmy, są i.i.d. □

Lemat 2.5.12. *Przypomnijmy: $W_{j,n}=\frac{1}{n}\sum_{i=0}^{n-1}\mathbb{1}_{\{X_i=j\}}$. Niech teraz $m_n=\inf\{k:\sum_{i=1}^{k}\gamma_i^{(j)}\geq n\}$. Wtedy:*

$$\frac{m_n-1}{\sum_{i=1}^{m_n}\gamma_i^{(j)}}\leq W_{j,n}\leq\frac{m_n}{\sum_{i=1}^{m_n-1}\gamma_i^{(j)}}$$

prawie na pewno.

Dowód. Z definicji γ wiemy, że jeśli $\sum_{i=1}^{k} \gamma_i^{(j)} < M$, to $\sum_{i=0}^{M-1} \mathbb{1}_{\{X_i=j\}} \geq k$. Podobnie, jeśli $\sum_{i=1}^{k} \gamma_i^{(j)} \geq M$, to $\sum_{i=0}^{M-1} \mathbb{1}_{\{X_i=j\}} < k$. Zatem z definicji m_n:

$$m_n - 1 \leq \sum_{i=0}^{n-1} \mathbb{1}_{\{X_i=j\}} \leq m_n \tag{1}$$

Podobnie z definicji m_n:

$$\sum_{i=1}^{m_n} \gamma_i^{(j)} \geq n \geq \sum_{i=1}^{m_n-1} \gamma_i^{(j)}$$

A więc:

$$\frac{1}{\sum_{i=1}^{m_n} \gamma_i^{(j)}} \leq \frac{1}{n} \leq \frac{1}{\sum_{i=1}^{m_n-1} \gamma_i^{(j)}} \tag{2}$$

Mnożąc (1) i (2) stronami dostajemy tezę. □

Ponieważ γ są skończone prawie na pewno, więc m_n są również skończone prawie na pewno, ponadto prawie na pewno $m_n \to \infty$ przy $n \to \infty$. W takim razie elementy ciągu

$$\frac{\sum_{i=1}^{m_n} \gamma_i^{(j)}}{m_n}$$

prawie na pewno pokrywają się z elementami ciągu:

$$\frac{\sum_{i=1}^{n} \gamma_i^{(j)}}{n},$$

a skoro jak już powiedziano $m_n \to \infty$, to prawie na pewno są równe w granicy. Stąd, z oczywistej własności $\frac{m_n}{m_n-1} \xrightarrow{p.n.} 1$ i z własności granicy prawie na pewno ($X_n \to X \Rightarrow \frac{1}{X_n} \to \frac{1}{X}$, jeśli tylko $X_n \neq 0$ prawie na pewno), widzimy, że skrajne strony nierówności z lematu dążą do $\frac{1}{\lambda_j}$ prawie na pewno. A więc z twierdzenia o trzech ciągach, również $W_{j,n} \to \frac{1}{\lambda_j}$ prawie na pewno.

Pozostaje dowieść, że $\pi_j = \frac{1}{\lambda_j}$. W tym celu wystarczy pokazać, że:

$$\lim_{n\to\infty} \mathbb{E}(W_{j,n}) = \pi_j$$

Dalej bowiem korzystamy z twierdzenia Lebesgue'a o zbieżności zmajoryzowanej – ponieważ $W_{j,n}$ są wspólnie ograniczone przez funkcję całkowalną $f \equiv 1$, zatem:

$$\lim_{n\to\infty} \mathbb{E}(W_{j,n}) = \mathbb{E}\left(\lim_{n\to\infty} W_{j,n}\right) = \mathbb{E}\left(\frac{1}{\lambda_j}\right) = \frac{1}{\lambda_j}$$

Kończymy zatem nasz dowód twierdzenia ergodycznego:

Twierdzenie 2.5.13.

$$\lim_{n\to\infty} \mathbb{E}(W_{j,n}) = \pi_j$$

Dowód. Zauważmy najpierw, że z już udowodnionego punktu (2) twierdzenia ergodycznego ($\boldsymbol{\nu}^{(n)}$ – rozkład JŁM w momencie n):

$$\lim_{n\to\infty} \mathbb{E}(\mathbb{1}_{\{X_n=j\}}) = \lim_{n\to\infty} \nu_j^{(n)} = \pi_j.$$

Ze znanego twierdzenia analizy, jeśli $a_n \xrightarrow{n\to\infty} a$, to także:

$$\frac{1}{n} \sum_{i=0}^{n-1} a_i \xrightarrow{n\to\infty} a,$$

co po zastosowaniu do ciągu $a_n := \mathbb{E}(\mathbb{1}_{\{X_n=j\}})$ i tożsamości (z liniowości wartości oczekiwanej) daje:

$$\mathbb{E}(W_{j,n}) = \frac{1}{n}\sum_{i=0}^{n-1}\mathbb{E}(\mathbb{1}_{\{X_i=j\}}) \to \pi_j$$

daje tezę i ostatecznie dowodzi punktu (3) twierdzenia ergodycznego. □

Zauważmy, że przy okazji dostajemy też ciekawą informację na temat rozkładu stacjonarnego, mianowicie $\pi_j = \frac{1}{\lambda_j}$, gdzie λ_j jest średnim czasem powrotu do j (tj. ile średnio czekamy na ponowną wizytę w j po jego odwiedzinach).

Punkt (3) twierdzenia ergodycznego pokazuje nam, że zbieżność do rozkładu jest *mocna* (co można było w jakiś sposób 'wyczuć' już na podstawie wykładniczego tempa zbieżności do rozkładu stacjonarnego omówionego w poprzednim podrozdziale). Gdybyśmy mieli tylko punkty (1) i (2) (czyli zbieżność według rozkładu), nie byłoby gwarancji na to, że zmienna losowa $W_{j,n}$ (średni czas przebywania w stanie j) nie *odchyla się* (mimo że coraz rzadziej, ale jednak nieskończenie wiele razy i dowolnie daleko) od π_j. *Mocna* zbieżność daje tę gwarancję – zapewnia nam, że od pewnego momentu średni czas przebywania w j jest niezmiennie bardzo bliski π_j (z pr. 1).
Nie ma to aż tak wielkiego znaczenia dla naszych problemów, jednak pokazuje, że JŁM swój cel – przybliżanie zmiennych losowych i.i.d. o rozkładzie $\boldsymbol{\pi}$ – realizuje naprawdę dokładnie. Z tego punktu wynika jeszcze jeden prosty wniosek.

Wniosek 2.5.14. *Niech $f : S \to \mathbb{R}$. Wtedy*

$$\frac{1}{n}\sum_{i=0}^{n-1} f(X_i) \overset{p.n.}{\to} \sum_{j\in S}\pi_j f(j)$$

Dowód. Wystarczy zauważyć, że:

$$f(X_i) = \sum_{j\in S} f(j)\mathbb{1}_{\{X_i=j\}}$$

Teza wynika z liniowości granicy prawie na pewno i punktu (3) twierdzenia ergodycznego. □

Warto na koniec zauważyć, że w istocie do punktów (1) i (3) twierdzenia ergodycznego wystarczyła nieredukowalność (X_n).
Ponieważ w rozwiązaniach będziemy operować na łańcuchach nieokresowych, przedstawię tylko szkic rozumowania. Zwróćmy uwagę, że w dowodzie tego, że $\gamma_n^{(s)},\ n = 2, 3...$ są i.i.d. nie korzystaliśmy z nieokresowości. Natomiast w dowodzie tego, że $\gamma_n^{(s)},\ n = 1, 2, 3...$ są skończone prawie na pewno nie trzeba z niej korzystać – wystarczy badać łańcuch 'ograniczony', do tych kroków, w których prawdopodobieństwo znalezienia się w danym stanie j nie jest zerowe. Podobnie dowodzi się istnienia momentów (jest jednak trochę szczegółów technicznych, które pomijam). Dalej dowód kontynuujemy jak dla przypadku nieokresowego. W efekcie dostajemy niezależną od rozkładu początkowego zbieżność $W_{n,j}$ do $\nu_j = \frac{1}{\lambda_j}$, takiego że $0 < \frac{1}{\lambda_j} < 1$. ν_j ze względu na definicję $W_{n,j}$ zadają rozkład. Jeśli JŁM jest nieokresowy, to rozkład ten jest jedynym stacjonarnym z udowodnionego wcześniej twierdzenia. Jeśli łańcuch natomiast jest okresowy, to o ile ma rozkład stacjonarny – jak łatwo dowieść – jest on właśnie równy $\boldsymbol{\nu}$. Pozostaje pokazać, że nieredukowalny, okresowy JŁM ma rozkład stacjonarny. Wystarczy zastosować prosty trick, żeby się o tym przekonać. Jeśli nieredukowalny JŁM o macierzy przejść $\mathbb{P} = (p_{ij})_{i,j\in S}$jest okresowy, to znaczy, że w szczególności $p_{ii} = 0$ dla każdego $i \in S$. Skonstruujmy nowy łańcuch z macierzą przejść $\mathbb{Q} = (q_{ij})_{i,j\in S}$, taki że dla wszystkich $i \in S$: $0 < q_{ii} = \varepsilon < 1$ oraz dla różnych i, j $q_{ij} = (1-\varepsilon)p_{ij}$, dla $i \neq s$. Łańcuch oczywiście nadal jest nieredukowalny (wszystkie prawdopodobieństwa przejść, które były dodatnie, są nadal dodatnie), ale nie jest już okresowy. Posiada zatem jedyny rozkład stacjonarny $\boldsymbol{\pi}$ – spełniający dla każdego j równanie balansu:

$$\pi_j = \sum_{i\in S, i\neq j}\pi_i(1-\varepsilon)p_{ij} + \varepsilon\pi_j$$

Przekształcamy równoważnie: po odjęciu stronami $\varepsilon\pi_j$ i podzieleniu przez $(1-\varepsilon)$ dostajemy równanie balansu oryginalnego łańcucha (X_n). Zatem istnieje jedyny rozkład stacjonarny okresowego łańcucha (X_n) i jest to $\boldsymbol{\pi}$.
Podobnie można pokazać, że omawiane w następnym podrozdziale centralne twierdzenie graniczne zachodzi także dla okresowych nieredukowalnych JŁM.

2.6 Centralne twierdzenie graniczne. Asymptotyczna wariancja

W poprzednich podrozdziałach udowodniliśmy twierdzenie ergodyczne, którego ostatni punkt był wersją mocnego prawa wielkich liczb dla jednorodnych łańcuchów Markowa. Jak się okazuje, dla jednorodnych łańcuchów Markowa zachodzi również wersja centralnego twierdzenia granicznego. Zostanie ono wyprowadzone i udowodnione w tym rozdziale. Zacznijmy od przypomnienia klasycznej wersji centralnego twierdzenia granicznego.

Twierdzenie 2.6.1 (CTG)**.** *Niech Z_n będzie ciągiem niezależnych zmiennych losowych o tym samym rozkładzie, skończonej wartości oczekiwanej μ i skończonej wariancji σ^2. Wtedy:*

$$\frac{\sum_{i=1}^{n} Z_i - n\mu}{\sqrt{n}} \xrightarrow{d} N(0,\sigma^2)$$

Naszym celem będzie udowodnienie podobnego twierdzenia dla jednorodnego łańcucha Markowa (X_n) i funkcji stanu $f : S \to \mathbb{R}$, tzn. powinna zachodzić zbieżność typu:

$$\frac{\sum_{i=0}^{n-1} f(X_i) - n\mu}{\sqrt{n}} \xrightarrow{d} N(0,\sigma^2)$$

dla pewnego σ i jak w (2.5.14) $\mu = \sum_{j\in S} \pi_j f(j)$.
Przyjmiemy podobną strategię jak w dowodzie wersji MPWL dla JŁM. To znaczy, będziemy próbować rozdzielić łańcuch na pewne niezależne części – taką rolę w dowodzie MPWL pełniły $\gamma_n^{(s)}$, tutaj użyjemy ich znowu. Dla wygody ustalmy s i na potrzeby tego rozumowania przyjmijmy $\gamma_n := \gamma_n^{(s)}$ i podobnie $\tau_n := \tau_n^{(s)}$. Oznaczmy następnie:

$$F_1 = \sum_{i=0}^{\tau_1} f(X_i)$$
$$F_n = \sum_{i=\tau_{n-1}+1}^{\tau_n} f(X_i) \quad \text{dla } n > 1$$

Oraz

$$G_{0,k} = \begin{cases} \sum_{i=0}^{k} f(X_i) & \text{dla } 0 \le k < \tau_1 \\ 0 & \text{dla } k > \tau_1 \end{cases}$$
$$G_{n,k} = \begin{cases} \sum_{i=\tau_n+1}^{\tau_n+k} f(X_i) & \text{dla } 1 \le k < \gamma_{n+1} \\ 0 & \text{dla } k = 0 \vee k \ge \gamma_{n+1} \end{cases} \qquad \text{dla } n > 0.$$

Twierdzenie 2.6.2. *Dla wszystkich n, k: F_n i $G_{n,k}$ są skończone prawie na pewno, ponadto wszystkie F_n mają skończoną wartość oczekiwaną i wariancję.*

Dowód. Zauważmy, że zachodzą nierówności:

$$|F_n| \le \gamma_n \cdot \max_{j\in S} |f(j)|$$
$$|G_{n,k}| \le \gamma_{n+1} \cdot \max_{j\in S} |f(j)|,$$

co dowodzi skończoności prawie na pewno, bo funkcja f jest ograniczona jako określona na skończonym zbiorze, natomiast γ_n są skończone prawie na pewno na mocy twierdzenia z poprzedniego podrozdziału. Podobnie, z pierwszej nierówności wynika istnienie skończonej wartości oczekiwanej, jako że γ_n ma skończoną wartość oczekiwaną ($\max_{j\in S}|f(j)|$ jest stałą). Również:

$$|F_n|^2 \leq \gamma_n^2 \cdot \max_{j\in S}|f(j)|^2$$

Z faktu z poprzedniego podrozdziału γ mają skończony drugi moment, a więc z powyższej nierówności również drugi moment F_n musi być skończony, a więc F_n ma skończoną wariancję. □

Twierdzenie 2.6.3. *$F_2, F_3, ...$ jest ciągiem i.i.d. zmiennych losowych.*

Dowód. Pokażemy najpierw, że rozkład F_n dla $n = 2, 3...$ nie zależy od n. Zwróćmy uwagę, że z dowodu twierdzenia o tym, że $\gamma_2, \gamma_3...$ mają ten sam rozkład można wywnioskować, że γ_n i τ_{n-1} są niezależne, bowiem warunkowy rozkład γ_n z warunkiem $\tau_{n-1} = t$ ewidentnie nie zależy od t. Weźmy zatem dowolny $A \in Bor(\mathbb{R})$, wtedy, korzystając z twierdzenia (2.1.4) i własności prawdopodobieństwa warunkowego:

$$P(F_n \in A) = \sum_{P(\tau_{n-1}=k,\gamma_n=l)>0} P(F_n \in A|\tau_{n-1}=k,\gamma_n=l)P(\tau_{n-1}=k,\gamma_n=l)$$

$$= \sum_{P(\tau_{n-1}=k,\gamma_n=l)>0} \frac{1}{P(\gamma_n=l)} P(F_n \in A,\gamma_n=l|\tau_{n-1}=k)P(\tau_{n-1}=k)P(\gamma_n=l)$$

$$= \sum_{\substack{P(\tau_{n-1}=k,\gamma_n=l)>0 \\ i_1,...i_{l-1}\in S\setminus\{s\},F_n\in A}} P(X_{k+l}=s, X_{k+l-1}=i_{l-1}..., X_{k+1}=i_1|X_k=s)P(\tau_{n-1}=k)$$

$$= \sum_{\substack{P(\tau_{n-1}=k,\gamma_n=l)>0 \\ i_1,...i_{l-1}\in S\setminus\{s\},F_n\in A}} p_{i_1 i_2}...p_{i_{l-1}i_l}P(\tau_{n-1}=k)$$

$$= \sum_{\substack{P(\gamma_n=l)>0 \\ i_1,...i_{l-1}\in S\setminus\{s\},F_n\in A}} \left(p_{i_1 i_2}...p_{i_{l-1}s}\left(\sum_{P(\tau_{n-1}=k)>0} P(\tau_{n-1}=k)\right)\right)$$

$$= \sum_{\substack{l:P(\gamma_n=l)>0 \\ i_1,...i_{l-1}\in S\setminus\{s\},F_n\in A}} p_{i_1 i_2}...p_{i_{l-1}s}.$$

Ale ponieważ $i_1...i_{l-1}$ to po prostu takie ciągi stanów, że

$$f(s) + \sum_{m=1}^{l-1} f(i_m) \in A$$

a więc nie zależą w żaden sposób od n, a γ_n mają ten sam rozkład dla $n = 2, 3, 4...$ Zatem widać, że rozkład F_n nie zależy od n – czyli zmienne mają ten sam rozkład. Zauważyć można także, że

$$P(F_n \in A|\gamma_n=l) = \frac{1}{P(\gamma_n=l)} \sum_{i_1,...i_{l-1}\in S\setminus\{s\},F_n\in A} p_{i_1 i_2}...p_{i_{l-1}s} \ (*),$$

gdzie sumujemy po ciągach jak poprzednio. Pozostaje pokazać niezależność zmiennych losowych w tym ciągu. Rozpiszmy więc:

$$P(F_n \in A_n, ..., F_2 \in A_2) = \sum_{\substack{l_1,l_2,...l_n: \\ P(\tau_1=l_1,\gamma_2=l_2,...\gamma_n=l_n)>0}} (P(F_n \in A_n, ..., F_2 \in A_2|\gamma_n=l_n...\gamma_2=l_2,\tau_1=l_1) \cdot P(\gamma_n=l_n,...\gamma_2=l_2,\tau_1=l_1)).$$

Skupmy się na pojedynczym składniku sumy, a konkretnie na części:

$$P(F_n \in A_n, ..., F_2 \in A_2 | \gamma_n = l_n ... \gamma_2 = l_2, \tau_1 = l_1).$$

Mamy:

$$\begin{aligned} P(F_n \in A_n, ..., F_2 \in A_2 | \gamma_n = l_n ... \gamma_2 = l_2, \tau_1 = l_1) = \\ P(F_n \in A_n | F_{n-1} \in A_{n-1}, ..., F_2 \in A_2, \gamma_n = l_n ... \gamma_2 = l_2, \tau_1 = l_1) \quad \cdot \\ P(F_{n-1} \in A_{n-1}, ..., F_2 \in A_2 | \gamma_n = l_n ... \gamma_2 = l_2, \tau_1 = l_1). \end{aligned}$$

Dalej z definicji F i γ, istnieje $B \subseteq S^{(l_1+l_2+l_3...+l_{n-1})}$, że

$$(X_0, ..., X_{l_1}, X_{l_1+1} ... X_{l_2+l_3...+l_{n-1}-1}) \in B$$

odpowiada zdarzeniu

$$E = \{F_{n-1} \in A_{n-1}, ..., F_2 \in A_2, \gamma_n = l_n ... \gamma_2 = l_2, \tau_1 = l_1\}.$$

Dla elementów E z definicji γ wynika, że $X_{l_1+l_2+l_3...+l_{n-1}} = s$. Oznaczmy dla wygody $l := l_1 + l_2 + l_3 ... + l_{n-1}$, korzystamy z twierdzenia (2.1.4) (dzięki istnieniu B) i definicji γ, a także podobnie jak poprzednio przyjmujemy, że $i_1 ... i_{l_n-1}$ to takie ciągi stanów, że

$$f(s) + \sum_{m=1}^{l_n-1} f(i_m) \in A_n,$$

otrzymując :

$$P(F_n \in A_n | F_{n-1} \in A_{n-1}, ..., F_2 \in A_2, \gamma_n = l_n ... \gamma_2 = l_2, \tau_1 = l_1) = P(F_n \in A_n | \gamma_n = l_n, X_l = s, E)$$

$$\begin{aligned} &= \frac{P(F_n \in A_n, \gamma_n = l_n | X_l = s, E)}{P(\gamma_n = l_n)} \\ &= \frac{P(F_n \in A_n, \gamma_n = l_n | X_l = s)}{P(\gamma_n = l_n)} \\ &= \frac{1}{P(\gamma_n = l_n)} \sum_{i_1, ... i_{l_n-1} \in S \setminus \{s\}, F_n \in A_n} P(X_{l+l_n} = s, X_{l+l_n-1} = i_{l_n-1}, ..., X_{l+1} = i_1 | X_l = s) \\ &= P(F_n \in A_n | \gamma_n = l_n) \quad \text{z (*)}. \end{aligned}$$

Przeprowadzając podobne rozumowanie, dostajemy również:

$$\begin{aligned} &P(F_{n-1} \in A_n | F_{n-2} \in A_{n-2}, ..., F_2 \in A_2, \gamma_n = l_n ... \gamma_2 = l_2) = \\ &P(F_{n-1} \in A_{n-1} | \gamma_{n-1} = l_{n-1}) \end{aligned}$$

i ogólnie:

$$\begin{aligned} &P(F_k \in A_k | F_{k-1} \in A_{k-1}, ..., F_2 \in A_2, \gamma_n = l_n ... \gamma_2 = l_2) = \\ &P(F_k \in A_k | \gamma_k = l_k) \end{aligned}$$

A więc z tożsamości:

$$P(A_n, ... A_1 | A_0) = P(A_n | A_{n-1}, ... A_0) P(A_{n-1} | A_{n-2} ... A_0) ... P(A_1 | A_0),$$

z wcześniejszych obliczeń, niezależności i skończoności γ dostajemy:

$$\begin{aligned} P(F_n \in A_n, ..., F_2 \in A_2) = \sum_{\substack{l_2, ... l_n: \\ P(\tau_1 = l_1, \gamma_2 = l_2, ... \gamma_n = l_n) > 0}} P(F_n \in A_2 | \gamma_n = l_n) ... P(F_2 \in A_2 | \gamma_2 = l_2) \\ P(\gamma_n = l_n) ... P(\gamma_2 = l_2) \\ = P(F_n \in A_n) P(F_{n-1} \in A_{n-1}) ... P(F_2 \in A_2) \end{aligned}$$

gdzie ostatnia równość wynika ze wzoru na prawdopodobieństwo całkowite i jest równoważna niezależności rozważanego ciągu zmiennych losowych. □

Choć dowód wydaje się skomplikowany, to technika jest w zasadzie taka sama, jak użyta do dowodu własności i.i.d. dla $\gamma_2, \gamma_3...$ Sama własność też jest intuicyjna – skoro czasy powrotu są i.i.d. to także to, co się dzieje pomiędzy kolejnymi z nich powinno być i.i.d.
Przed ostatecznym dowodem CTG dla JŁM zostaje nam jeszcze jeden lemat.

Lemat 2.6.4. *Dla F_i, $i = 2, 3, 4...$ jest*

$$\mathbb{E}(F_i) = \frac{\mu}{\pi_s} = \frac{1}{\pi_s} \sum_{j \in S} \pi_j f(j).$$

Dowód. Niech $m_n = \inf\{k : \sum_{i=0}^{k} \gamma_i^{(s)} \geq n\}$. Wtedy:

$$\mu = \sum_{j \in S} \pi_j f(j) = \lim_{n \to \infty} \frac{1}{n} \sum_{k=0}^{n-1} f(X_k) = \lim_{n \to \infty} \frac{m_n}{n} \cdot \frac{\sum_{k=1}^{m_n} F_k}{m_n},$$

gdzie równości są prawie na pewno. Podobnie jak wcześniej F_1 jest skończona prawie na pewno, $\frac{m_n}{n} \to \pi_s$ prawie na pewno, m_n są wszystkie skończone prawie na pewno i dążą do nieskończoności przy n dążącym do nieskończoności. Pamiętając, że $F_2, F_3...$ są i.i.d. i korzystając z arytmetyki granic dostajemy:

$$\mu = \pi_s \lim_{n \to \infty} \left(\frac{F_1}{m_n} + \frac{m_n - 1}{m_n} \frac{\sum_{k=2}^{m_n} F_k}{m_n - 1} \right).$$

Oczywiście $\frac{F_1}{m_n}$ dąży do 0 prawie na pewno. $\frac{m_n - 1}{m_n}$ dąży prawie na pewno do 1, natomiast elementy ciągu

$$\frac{\sum_{k=2}^{m_n} F_k}{m_n - 1}$$

prawie na pewno pokrywają się z elementami

$$\frac{\sum_{k=2}^{n} F_k}{n - 1},$$

a te ostatnie na mocy zwykłego MPWL dążą do $\mathbb{E}[F_2]$ prawie na pewno. Ostatecznie więc:

$$\mu = \pi_s \cdot \mathbb{E}[F_2],$$

A ponieważ $\mathbb{E}[F_2] = \mathbb{E}[F_3]...$, więc:

$$\mathbb{E}[F_i] = \frac{\mu}{\pi_s}$$

dla $i = 2, 3, 4...$, co należało udowodnić. □

Możemy teraz dowieść CTG dla łańcuchów Markowa.

Twierdzenie 2.6.5 (CTG dla JŁM)**.** *Niech (X_n) będzie nieredukowalnym i nieokresowym JŁM, wtedy:*

$$\frac{\sum_{i=0}^{n-1} f(X_i) - n\mu}{\sqrt{n}} \xrightarrow{d} N(0, \sigma^2)$$

dla pewnego σ oraz $\mu = \sum_{j \in S} \pi_j f(j)$.

Dowód. Ustalmy m_n - ze względów technicznych podobne, ale nieco różniące się od tego, którego używaliśmy wcześniej: $m_n = \inf\{k : \sum_{i=0}^{k} \gamma_i^{(s)} > n\}$. Dostajemy wtedy:

$$\begin{aligned}
\frac{\sum_{i=0}^{n-1} f(X_i) - n\mu}{\sqrt{n}} &= \frac{\sum_{i=1}^{m_n-1} F_i + G_{m_n-1,n+1-m_n} - \frac{n}{m_n-2}(m_n-2)\mu}{\sqrt{n}} \\
&= \frac{F_1 + \sum_{i=2}^{m_n-1} F_i + G_{m_n-1,n+1-m_n}}{\sqrt{n}} \\
&= \frac{F_1 + G_{m_n-1,n+1-m_n}}{\sqrt{n}} + \sqrt{\frac{m_n-2}{n}} \cdot \frac{\sum_{i=2}^{m_n-1} F_i - \frac{n}{m_n-2}(m_n-2)\mu}{\sqrt{m_n-2}}
\end{aligned}$$

Ze względu na własności m_n udowodnione przy okazji MPWL, wiemy, że:

$$\lim_{n\to\infty} \frac{m_n - 2}{n} = \pi_s \qquad p.n.,$$

więc z arytmetyki granic:

$$\lim_{n\to\infty} \frac{\sum_{i=2}^{m_n-1} F_i - \frac{n}{m_n-2}(m_n-2)\mu}{\sqrt{m_n-2}} = \lim_{n\to\infty} \frac{\sum_{i=2}^{n-1} F_i - (n-2)\frac{\mu}{\pi_s}}{\sqrt{n-2}}$$

prawie na pewno. Ze względu na lemat (2.6.4) oraz udowodnioną wcześniej skończoność wariancji F_i dla prawej strony zachodzi klasyczne CTG. Tak więc również lewa strona zbiega wg rozkładu do tej samej zmiennej losowej. Ponadto F i G są skończone prawie na pewno, więc pierwszy składnik dąży do 0 prawie na pewno (czyli również wg rozkładu). Toteż ostatecznie na mocy klasycznego CTG:

$$\frac{\sum_{i=0}^{n-1} f(X_i) - n\mu}{\sqrt{n}} \xrightarrow{d} N(0, \pi_s Var[F_2])$$

□

Zwróćmy uwagę, że skoro zachodzi zbieżność wg rozkładu, to również zbiega ciąg wariancji:

$$Var_f^{(n)} = \frac{1}{n} Var\left(\sum_{i=0}^{n-1} f(X_i)\right) \to \pi_s Var[F_2]),$$

zatem możemy wprowadzić pojęcie asymptotycznej wariancji.

Definicja 2.6.6. *Asymptotyczną wariancją funkcji f dla JŁM $\{X_n\}$ o macierzy przejścia $\mathbb{P}$ nazwiemy:*

$$Var_f(\mathbb{P}) = \lim_{n\to\infty} \frac{1}{n} Var\left(\sum_{i=0}^{n-1} f(X_i)\right)$$

Zauważmy, że rozkład F_2 nie zależał od rozkładu początkowego (a jedynie od macierzy przejścia i stanu s). Zatem skoro

$$Var_f = \pi_s Var[F_2],$$

to wariancja asymptotyczna nie zależy od rozkładu początkowego. Ponadto

$$\pi_s Var[F_2]$$

nie zależy od s.

Centralne twierdzenie graniczne jest kolejnym argumentem na to, że jednorodny łańcuch Markowa jest pod wieloma względami bardzo podobny do ciągu niezależnych zmiennych losowych o rozkładzie $\boldsymbol{\pi}$. To podobieństwo, w połączeniu z efektywnymi metodami symulacji, jest głównym powodem, dla którego algorytmy MCMC (Markov Chain Monte Carlo) są tak przydatne i mogą nam posłużyć do rozwiązania postawionych w tej pracy problemów.

Ponadto widać, że zachodzi asymptotyczna zbieżność, więc ma sens analiza asymptotycznych własności (np. wariancji).

2.7 Podsumowanie

Głównym wynikiem tego rozdziału jest twierdzenie ergodyczne, które daje nam intuicję, jak znaleźć stan, dla którego pewna funkcja (oddająca jakoś rzeczywistość) $f : S \to \mathbb{R}$ przyjmuje maksymalną wartość. Centralne twierdzenie graniczne ponadto daje uzasadnienie dla badania pewnych asymptotycznych własności.

Jak widać z twierdzenia ergodycznego π_j zadaje średni czas przebywania w stanie j w długim okresie. Powinniśmy zatem skonstruować (zasymulować) ergodyczny JŁM, taki że jego rozkład stacjonarny jest proporcjonalny do f (czyli $\pi_j = c \cdot f(j)$ dla pewnej stałej c). Wówczas dla odpowiednio dużego n stan maksymalizujący f zostanie odwiedzony z dużym prawdopodobieństwem – ogólnie stany z dużą wartością funkcji f będą odwiedzane często, a pozostałe rzadko, co jest nam oczywiście na rękę.

Taką strategię przyjmiemy dla dekodowania zaszyfrowanego tekstu i podobną do rozwiązania problemu komiwojażera. Potrzebujemy więc dwóch rzeczy: konstrukcji JŁM o proporcjonalnym do zadanej funkcji stanu rozkładzie stacjonarnym i odpowiedniej funkcji stanu. Zaczniemy od opisania w następnym rozdziale, jak skonstruować pożądany JŁM przy pomocy metod Monte Carlo, dobieraniem odpowiedniej funkcji stanu zajmiemy się w rozdziałach dotyczących konkretnych problemów. Na potem przyda się jeszcze jeden lemat:

Lemat 2.7.1. *Jeśli dla jednorodnego łańcucha Markowa o macierzy przejść $\mathbb{P} = (p_{ij})_{i,j \in S}$ rozkład $\boldsymbol{\pi}$ spełnia*

$$\pi_j p_{ji} = \pi_i p_{ij}, \forall i, j \in S$$

to $\boldsymbol{\pi}$ jest rozkładem stacjonarnym.

Dowód. Dodając stronami tożsamości dla wszystkich $i \in S$ dostajemy:

$$\pi_j \sum_{i \in S} p_{ji} = \pi_j \cdot 1 = \pi_j = \sum_{i \in S} \pi_i p_{ij}$$

□

Równości z lematu noszą nazwę *detailed balance equations* i jak się okazuje są warunkiem koniecznym i wystarczającym na tzw. *odwracalność* stacjonarnego JŁM $\{X_n\}$, tzn. dla każdego $n = 0, 1, 2, ...$ i stacjonarnego JŁM $\{X_n\}$ równość rozkładów:

$$(X_0, X_1, ..., X_n) \stackrel{d}{=} (X_n, X_{n-1}, ..., X_0)$$

(co jest definicją odwracalności) zachodzi wtedy i tylko wtedy, gdy *detailed balance equations* są spełnione. Możemy przeprowadzić krótki dowód tego faktu Najpierw załóżmy, że łańcuch jest odwracalny. Wtedy dla dowolnych $i, j \in S$. Wtedy $P(X_0 = i, X_1 = j) = \pi_i p_{ij} = P(X_1 = i, X_0 = j) = \pi_j p_{ji}$, zatem zachodzą *detailed balance equations*. Następnie załóżmy, że zachodzą *detailed balance equations*. Wtedy dla $i_0, ..., i_n \in S$:

$$\begin{aligned} P(X_0 = i_0, X_1 = i_1, ...X_n = i_n) &= \pi_{i_0} p_{i_0 i_1} ... p_{i_{n-1} i_n} = p_{i_1 i_0} \pi_{i_1} p_{i_1 i_2} ... p_{i_{n-1} i_n} = ... \\ &= p_{i_1 i_0} p_{i_2 i_1} ... p_{i_{n-2} i_{n-1}} \pi_{n-1} p_{i_{n-1} i_n} = p_{i_1 i_0} p_{i_2 i_1} ... p_{i_n i_{n-1}} \pi_{i_n} \\ &= P(X_n = i_0, X_{n-1} = i_1, ...X_0 = i_n) \end{aligned}$$

co oznacza, że łańcuch jest odwracalny. □

3 Metody Monte Carlo

W rozdziale tym zajmę się metodami generowania łańcuchów Markowa o zadanych własnościach, a także spróbuję przybliżyć własności w ten sposób wygenerowanych łańcuchów.

3.1 Wprowadzenie

Czym są klasyczne metody Monte Carlo? W uproszczeniu jest to szukanie rozwiązania (być może przybliżonego) danego problemu przy użyciu ciągu niezależnych zmiennych losowych (pseudolosowych).
Oryginalną motywacją dla metod Monte Carlo było obliczanie pewnych wartości, tudzież symulowanie modeli, które są zbyt skomplikowane (lub niemożliwe) – np. zajmuje to zbyt wiele czasu – do obliczenia za pomocą metod analitycznych czy klasycznych metod numerycznych.

Wyobraźmy sobie płaską figurę geometryczną $\mathcal{F}$ położoną wewnątrz kwadratu jednostkowego opisaną skomplikowanym wzorem. Niech wzór będzie na tyle skomplikowany, że zarówno użycie metod analitycznych jak i numerycznych jest niepraktyczne/niewygodne/niemożliwe przez nadmierne skomplikowanie czy zbyt długi czas obliczeń, jednak dla dowolnego punktu z kwadratu sprawdzenie, czy należy on do figury jest proste (jest to jakaś wersja problemu P vs NP). Losujmy kolejno jednostajnie punkty P_n z kwadratu. Zgodnie z klasycznym prawdopodobieństwem geometrycznym:

$$P(P_n \in \mathcal{F}) = \mathbb{E}\mathbf{1}_{\mathcal{F}}(P_n) = Pole(\mathcal{F}).$$

Widać więc, że generujemy w ten sposób ciąg i.i.d. zmiennych losowych $\mathbf{1}_{\mathcal{F}}(P_n)$ o rozkładzie dwupunktowym

$$P(\mathbf{1}_{\mathcal{F}} = 1) = Pole(\mathcal{F}), P(\mathbf{1}_{\mathcal{F}} = 0) = 1 - Pole(\mathcal{F})$$

Zatem z prawa wielkich liczb:

$$\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\mathcal{F}}(P_n) \overset{p.n.}{\to} Pole(\mathcal{F})$$

Stąd mamy algorytm (Monte Carlo) na przybliżenie tego pola, wystarczy losować kolejno punkty z kwadratu i liczyć proporcję tych z wnętrza figury do wszystkich wylosowanych punktów. Oczywiście tempo zbieżności takiego algorytmu wymaga analizy (ponadto trzeba wziąć pod uwagę jakość generatora liczb pseudolosowych) – widać jednak, że wielką jego zaletą jest prostota – zarówno idei, jak i implementacji.

Na tym przykładzie widać dwa najczęstsze zastosowania metod Monte Carlo:
(1) symulacja ciągu zmiennych losowych o pewnym, nieznanym *wprost*, rozkładzie – tu naszą zmienną było $\mathcal{F}(P)$ o nieznanym rozkładzie $(Pole(\mathcal{F}), 1 - Pole(\mathcal{F}))$ a Ω tworzyły punkty P z kwadratu.
(2) Przybliżenie wartości oczekiwanej pewnej zmiennej losowej za pomocą symulacji.

Co do idei podejście do rozwiązania problemów w tej pracy jest podobne, jednak jak zostanie opisane w kolejnym podrozdziale – rozkłady, a także przestrzenie probabilistyczne, które się tu pojawiają, są dużo bardziej skomplikowane niż kwadrat jednostkowy i $\mathbf{1}_{\mathcal{F}}$. Stąd użycie klasycznych metod Monte Carlo (z niezależnym losowaniem zmiennych z rozkładu) nie jest możliwe ze względów pamięciowych i obliczeniowych. Z pomocą przychodzą nam metody Markov Chain Monte Carlo (MCMC, próbkowanie Monte Carlo łańcuchami Markowa).

3.2 Metody MCMC. Dlaczego ich potrzebujemy?

Z pewnych powodów, w większości przypadków, do problemów dekodowania i komiwojażera, co szerzej omówimy w odpowiadających im rozdziałach – nie możemy (albo przynajmniej jest to pod pewnym względem nieoptymalne) użyć metod analitycznych, gradientowych ani numerycznych (albo funkcja, którą chcemy maksymalizować, ma *dziwną* i przede wszystkim nieznaną strukturę,

albo przestrzeń, po której się poruszamy jest *dziwna*, albo to i to – co skreśla te metody). Pozostaje nam więc podejście probabilistyczne (może ono przyjąć różne formy), ze względu na specyfikę zwłaszcza problemu dekodowania obieramy podejście Monte Carlo, tj. chcemy losować kolejne propozycje rozwiązań wg rozkładu takiego, że im rozwiązanie (stan w S) lepsze, tym większe powinno mieć prawdopodobieństwo wystąpienia. Chcemy wprowadzić pewną funkcję stanu (oddającą rzeczywistość, czyli np. częstość występowania danej sekwencji znaków w języku angielskim), taką że im większa jej wartość, tym stan jest *lepszy*. Chcemy znaleźć stan, przy którym ta funkcja przyjmuje maksimum. Toteż im większa wartość funkcji stanu, tym większe ma być prawdopodobieństwo wylosowania. Idea jest zatem taka, że losujemy kolejno zmienne wg rozkładu, którego nie znamy, jednak wiemy, że jest proporcjonalny do wartości funkcji, którą umiemy obliczać. Problem jest taki, że w przeciwieństwie do przykładu z figurą rozkład ten jest dużo bardziej skomplikowany, moc przestrzeni stanów jest bardzo duża – – wykładniczego rzędu – $O(n!)$ lub $O(k^n)$ (k, n rzędu kilkudziesięciu do kilkuset). Ideą jest losować kolejno z tego rozkładu i jeśli wartość funkcji dla *poprawnego* rozwiązania jest duża – powinniśmy w pewnym rozsądnym czasie to rozwiązanie uzyskać (albo rozwiązanie w jakiś sposób *zbliżone* do poprawnego). Gdybyśmy jednak to chcieli zrobić podobnie jak z figurą, musielibyśmy przechowywać pełną informację o rozkładzie (w przypadku figury, tą informacją jest jej równanie), w naszych przypadkach, jak już powiedzieliśmy, rozkłady są skomplikowane i skupione na dużej przestrzeni stanów. Tak że samo przechowywanie informacji o rozkładzie wymagałoby przynajmniej wykładniczej pamięci (i wykładniczego czasu obliczeń). Stąd próby losowania zmiennej losowej o *dokładnie* tym rozkładzie są skazane na niepowodzenie ze względu na ograniczenia komputera (na dużej przestrzeni stanów umiemy losować *wprost* tylko z pewnych specyficznych i w pełni znanych rozkładów np. jednostajnego)
W dowolnym rozwiązaniu opierającym się na podejściu probabilistycznym należy zatem w jakiś sposób zredukować problem do niższego wymiaru. Będziemy to robić, używając metod MCMC. Dzięki tym metodom jesteśmy w stanie świetnie przybliżać dany rozkład, mając jedynie stan początkowy i informację o tym, jak obliczać pewną funkcję (w wielomianowym, ew. pseudowielomianowym czasie) proporcjonalną do rozkładu, nie mając pełnej informacji o rozkładzie (chociażby nie znając stałej normalizującej, której należy użyć, by z funkcji dostać rozkład). Jest to ogólna nazwa metod, które (a) tak jak w klasycznej metodzie Monte Carlo komputerowo symulują pewne zdarzenia losowe (tak jak losowanie punktu z kwadratu w przykładzie) (b) następnie używają tych zasymulowanych zdarzeń do generacji łańcucha Markowa o pożądanym rozkładzie stacjonarnym. Z wcześniej wyprowadzonych twierdzeń o zbieżności takiego łańcucha do rozkładu stacjonarnego, wiemy, że możemy badać własności związane z rozkładem stacjonarnym, badając zachowanie tego łańcucha.

3.3 Algorytm Metropolisa i algorytm Metropolisa-Hastingsa

Z pewnych powodów, które opiszę w następnym podrozdziale, najlepszymi algorytmami Monte Carlo do generacji łańcucha Markowa o pożądanym rozkładzie stacjonarnym są algorytm Metropolisa i jego uogólnienie - algorytm Metropolisa-Hastingsa. Zacznijmy od algorytmu Metropolisa.

Załóżmy, że chcemy zasymulować jednorodny łańcuch Markowa na przestrzeni stanów S, tak żeby jego rozkład stacjonarny wynosił $\boldsymbol{\pi}$, gdzie $\pi_i > 0$ dla każdego $i \in S$. Załóżmy najpierw, że mamy do dyspozycji pewną macierz przejścia na S, nieredukowalną i nieokresową macierz stochastyczną $\mathbb{Q}$ - zwaną *macierzą generującą kandydatów* (nazwa bierze się stąd, że na jej podstawie losujemy *kandydata*, który następnie albo jest akceptowany jako następny stan, albo łańcuch zostaje w swoim aktualnym stanie). Macierz $\mathbb{Q}$ nie musi (i najprawdopodobniej tego nie robi) zadawać łańcucha z rozkładem stacjonarnym $\boldsymbol{\pi}$. Na początek, co w wielu przypadkach jest naturalne, załóżmy, że $\mathbb{Q} = (q_{ij})_{i,j\in S}$ jest symetryczna, czyli $q_{ij} = q_{ji}$ dla $i, j \in S$. Rozważmy macierz $\mathbb{P} = (p_{ij})_{i,j\in S}$ zależną od $\mathbb{Q}$ i $\boldsymbol{\pi}$ w następujący sposób:

$$p_{ij} = \begin{cases} q_{ij} \min(1, \frac{\pi_j}{\pi_i}) & \text{dla } i \neq j \\ 1 - \sum\limits_{j \in S \setminus \{i\}} p_{ij} & \text{dla } i = j \end{cases}$$

Twierdzenie 3.3.1. *Tak zdefiniowana $\mathbb{P}$ jest stochastyczna, nieredukowalna i nieokresowa i zadaje łańcuch o rozkładzie stacjonarnym $\boldsymbol{\pi}$.*

Dowód. Że wyrazy $\mathbb{P}$ sumują się do 1, wynika wprost z definicji. Ponadto dla każdego $i \neq j$: $p_{ij} \geq 0$

jako iloczyn liczb nieujemnych. Pozostaje pokazać, że dla każdego $i \in S$ $p_{ii} \geq 0$, ale:

$$p_{ii} = 1 - \sum_{j \in S \setminus \{i\}} p_{ij} = 1 - \sum_{j \in S \setminus \{i\}} q_{ij} \min(1, \frac{\pi_j}{\pi_i}) \geq 1 - \sum_{j \in S \setminus \{i\}} q_{ij} = q_{ii} \geq 0,$$

jako że $\mathbb{Q}$ jest macierzą stochastyczną. Zatem $\mathbb{P}$ jest stochastyczna. Ponadto z założenia o π mamy $0 < \min(1, \frac{\pi_j}{\pi_i})$, stąd i z poprzedniej nierówności widać, że $p_{ij} > 0$, jeśli tylko $q_{ij} > 0$, zatem własności nieokresowości i nieredukowalności są zachowane. Pokażemy następnie, że π spełnia równanie balansu dla tej macierzy. Istotnie, zauważmy, że dla dowolnych $i, j \in S$, $i \neq j$:

$$\pi_j p_{ji} = \pi_j q_{ji} \min(1, \frac{\pi_i}{\pi_j}) = q_{ji} \min(\pi_j, \pi_i) = q_{ji} \pi_i \min(\frac{\pi_j}{\pi_i}, 1) = \pi_i q_{ij} \min(\frac{\pi_j}{\pi_i}, 1) = \pi_i p_{ij},$$

gdzie przedostatnia równość wynika z symetrii macierzy $\mathbb{Q}$. Ponadto w oczywisty sposób równość $\pi_i p_{ij} = \pi_j p_{ji}$ zachodzi dla $i = j$ Zatem z (2.7.1) równanie balansu jest spełnione, stąd $\mathbb{P}$ tak zdefiniowana zadaje JŁM o rozkładzie stacjonarnym $\boldsymbol{\pi}$. □

Z powyższej konstrukcji wynika algorytm Metropolisa (zaproponowany pierwszy raz w 1953 w [7]) generacji JŁM (X_n) o rozkładzie stacjonarnym $\boldsymbol{\pi}$, gdy mamy daną macierz kandydatów $\mathbb{Q}$, informację jak obliczać $\frac{\pi_i}{\pi_j}$ i generator i.i.d. prób z $U[0,1]$:

Algorytm 1: Algorytm Metropolisa

Zacznij w dowolnie wybranym stanie $X_0 := i \in S$
dla: $n = 0, 1, 2, 3...$
  wylosuj Z – kandydata na nowy stan zgodnie z rozkładem $q_{X_n \cdot}$ (np. może być $Z := \varphi_{\mathbb{Q}}(U, X_n)$ dla $U \sim U[0,1]$);
  wylosuj $V \sim U[0,1]$;
  jeśli: $V \leq \min(1, \frac{\pi_Z}{\pi_{X_n}})$
    $X_{n+1} := Z$;
  w przeciwnym razie:
    $X_{n+1} := X_n$;

Łatwo sprawdzić, że taki algorytm generuje łańcuch o macierzy przejścia $\mathbb{P}$ opisanej wcześniej.

Teraz możemy pozbyć się założenia o symetryczności macierzy $\mathbb{Q}$ – będzie to algorytm Metropolisa-Hastingsa.

Tym razem dysponujemy stochastyczną macierzą generującą kandydatów $\mathbb{Q}$, która niekoniecznie jest symetryczna, natomiast nadal jest nieredukowalna i nieokresowa, ponadto $q_{ij} > 0 \Leftrightarrow q_{ji} > 0$. Rozważmy macierz przejścia $\mathbb{P} = (p_{ij})_{i,j \in S}$ o bardzo podobnym wzorze jak w przypadku algorytmu Metropolisa:

$$p_{ij} = \begin{cases} q_{ij} \min(1, \frac{\pi_j q_{ji}}{\pi_i q_{ij}}) & \text{dla } i \neq j \text{ i } q_{ij} \neq 0 \\ 0 & \text{dla } i \neq j \text{ i } q_{ij} = 0 \\ 1 - \sum\limits_{j \in S \setminus \{i\}} p_{ij} & \text{dla } i = j \end{cases}$$

Zachodzi analogiczne twierdzenie jak w poprzednim przypadku.

Twierdzenie 3.3.2. *Tak zdefiniowana $\mathbb{P}$ jest stochastyczna, nieredukowalna i nieokresowa i zadaje łańcuch o rozkładzie stacjonarnym $\boldsymbol{\pi}$.*

Dowód. Stochastyczności dowodzimy tak jak poprzednio: wyrazy w wierszu sumują się do jeden, ponadto dla $i \neq j$ jest $0 \leq p_{ij} \leq q_{ij}$, zatem wszystkie wyrazy są nieujemne. Nieredukowalność i nieokresowość wynikają podobnie jak poprzednio, z tego, że $p_{ij} > 0 \Leftrightarrow q_{ij} > 0$. Dalej wystarczy sprawdzić, że są spełnione założenia lematu (2.7.1). Dla $i = j$ oraz gdy $q_{ij} = q_{ji} = 0$ jest to oczywiste, w przeciwnym razie:

$$\pi_j p_{ji} = \pi_j q_{ji} \min(1, \frac{\pi_i q_{ij}}{\pi_j q_{ji}}) = \min(\pi_j q_{ji}, \pi_i q_{ij}) = \pi_i q_{ij} \min(\frac{\pi_j q_{ji}}{\pi_i q_{ij}}, 1) = \pi_i p_{ij}.$$

Zatem równanie balansu jest spełnione, więc $\boldsymbol{\pi}$ jest rozkładem stacjonarnym. □

W takim razie możemy zapisać algorytm [8]:

Algorytm 2: Algorytm Metropolisa-Hastingsa

Zacznij w dowolnie wybranym stanie $X_0 := i \in S$;
dla: $n = 0, 1, 2, 3...$
  wylosuj Z – kandydata na nowy stan zgodnie z rozkładem $q_{X_n \cdot}$ (np. może być $Z := \varphi_{\mathbb{Q}}(U, X_n)$ dla $U \sim U[0,1]$);
  wylosuj $V \sim U[0,1]$;
  jeśli: $V \leq \min(1, \frac{\pi_Z q_{X_n Z}}{\pi_{X_n} q_{Z X_n}})$
    $X_{n+1} := Z$;
  w przeciwnym razie:
    $X_{n+1} := X_n$;

Również łatwo sprawdzamy, że otrzymujemy w ten sposób łańcuch zadany macierzą $\mathbb{P}$ z równania. Uwaga: zauważmy, że do nieokresowości łańcucha wynikowego jest wystarczająca, ale nie jest konieczna nieokresowość macierzy $\mathbb{Q}$, więc zwykle w zastosowaniach przejmujemy się tylko nieokresowością $\mathbb{P}$. Dowody nieredukowalności i spełniania równań balansu nie zależą od nieokresowości, tak więc w praktyce $\mathbb{Q}$ może nie być nieokresowa.
Zwróćmy uwagę, że proces przejścia naturalnie rozdziela się na wybór kandydata (przy pomocy *macierzy generującej*), a następnie akceptację lub odrzucenie i pozostanie przy aktualnym stanie. Można zatem zdefiniować *prawdopodobieństwo akceptacji* α_{ij}, czyli prawdopodobieństwo przejścia z i do j ($i \neq j$) pod warunkiem, że j został wylosowany jako kandydat. Dla algorytmu Metropolisa mamy:

$$\alpha_{ij} = \min(1, \frac{\pi_j}{\pi_i}).$$

Natomiast dla uogólnienia Metropolisa-Hastingsa (tam, gdzie $q_{ij} \neq 0$, w przeciwnym razie przyjmujemy $\alpha_{ij} = 1$):

$$\alpha_{ij} = \min(1, \frac{\pi_j q_{ji}}{\pi_i q_{ij}})$$

I prawdopodobieństwo przejścia $p_{ij} = q_{ij}\alpha_{ij}$ gdy $i \neq j$ oraz $p_{ii} = 1 - \sum_{j \in S, j \neq i} p_{ij}$.

Ogólnie za [8] możemy wprowadzić pewną rodzinę algorytmów, czyli równoważnie stochastycznych, nieredukowalnych i nieokresowych macierzy przejść $\mathbb{P}$ o stacjonarnym rozkładzie $\boldsymbol{\pi}$, wyprowadzonych na podstawie $\mathbb{Q}$ o założeniach jak w algorytmie Metropolisa-Hastingsa (zgodnie z uwagą możemy pominąć nieokresowość). Sprowadza się to do ustalenia warunków na $(\alpha_{ij})_{i,j \in S, i \neq j}$, gdzie $0 < \alpha_{ij} \leq 1$ (chcemy, żeby każdy wylosowany *kandydat* miał niezerowe szanse na zostanie stanem w następnym kroku). Mamy więc:

$$p_{ij} = q_{ij}\alpha_{ij}.$$

I ustalamy:

$$0 \leq \alpha_{ij} = \frac{s_{ij}}{1 + t_{ij}} \leq 1,$$

gdzie s_{ij} są zadane pewną symetryczną funkcją ($s_{ij} = s_{ji}$) $S \times S \to \mathbb{R}_{\geq 0}$, natomiast

$$t_{ij} = \frac{\pi_i q_{ij}}{\pi_j q_{ji}},$$

tam gdzie $q_{ij} \neq 0$ oraz $t_{ij} = 0$ w przeciwnym razie.

Ostatecznie więc jest to rodzina symetrycznych nieujemnych funkcji s_{ij}, takich, że wynikowe α_{ij} zadają $0 < p_{ij} \leq 1$ o pożądanych własnościach.

Po pierwsze zauważmy, że α w wersji z algorytmu Metropolisa-Hastingsa należy do tej rodziny. Rzeczywiście, przyjmując dla $i, j \in S,\ i \neq j$, że $q_{ij} \neq 0$:

$$s_{ij} = 1 + \min(t_{ij}, t_{ji}) = 1 + \min\left(\frac{\pi_i q_{ij}}{\pi_j q_{ji}}, \frac{\pi_j q_{ji}}{\pi_i q_{ij}}\right).$$

i $s_{ij} = 1$ tam, gdzie $q_{ij} = 0$ mamy:

$$\alpha_{ij} = 1.$$

Dla $i \neq j \in S$, że $q_{ij} \neq 0$ przyjmijmy natomiast bez straty ogólności:

$$\min\left(\frac{\pi_i q_{ij}}{\pi_j q_{ji}}, \frac{\pi_j q_{ji}}{\pi_i q_{ij}}\right) = \frac{\pi_i q_{ij}}{\pi_j q_{ji}}.$$

Wtedy:

$$0 < \frac{\pi_i q_{ij}}{\pi_j q_{ji}} \leq 1 \leq \frac{\pi_j q_{ji}}{\pi_i q_{ij}}.$$

Zatem:

$$\alpha_{ij} = \frac{1 + \frac{\pi_i q_{ij}}{\pi_j q_{ji}}}{1 + \frac{\pi_i q_{ij}}{\pi_j q_{ji}}} = 1 = \min(1, \frac{\pi_j q_{ji}}{\pi_i q_{ij}})$$

oraz

$$\alpha_{ji} = \frac{1 + \frac{\pi_i q_{ij}}{\pi_j q_{ji}}}{1 + \frac{\pi_j q_{ji}}{\pi_i q_{ij}}} = \frac{\frac{\pi_i q_{ij} + \pi_j q_{ji}}{\pi_j q_{ji}}}{\frac{\pi_i q_{ij} + \pi_j q_{ji}}{\pi_i q_{ij}}} = \frac{\pi_i q_{ij}}{\pi_j q_{ji}} = \min(1, \frac{\pi_i q_{ij}}{\pi_j q_{ji}}).$$

Oczywiście tak zdefiniowana s_{ij} jest symetryczna, zatem algorytm Metropolisa-Hastingsa (i Metropolisa) należy do tej rodziny. Ponadto można pokazać, że dla wszystkich $\boldsymbol{\alpha}$ z tej rodziny, jeśli tylko dla wszystkich $i \neq j \in S$ zachodzi $0 < \alpha_{ij} \leq 1$, to wynikowa macierz $\mathbb{P}$ jest nieredukowalna i ma rozkład stacjonarny $\boldsymbol{\pi}$. Istotnie, ponieważ $p_{ij} = \alpha_{ij} q_{ij}$ są dodatnie, jeśli tylko q_{ij} są dodatnie, więc nieredukowalność jest zachowana. Natomiast dzięki definicji $\boldsymbol{\alpha}$ jest spełnione *detailed balance equation* (2.7.1): dla $i = j$, jest to oczywiste, jeśli $q_{ij} = 0$, po obu stronach równości dostajemy 0, pozostaje więc przypadek, gdy $q_{ij} > 0$:

$$\pi_j p_{ji} = \pi_j q_{ji} \frac{s_{ji}}{1 + \frac{\pi_j q_{ji}}{\pi_i q_{ij}}} = \frac{s_{ji}}{\frac{1}{\pi_j q_{ji}} + \frac{1}{\pi_i q_{ij}}} = \frac{s_{ij}}{\frac{1}{\pi_j q_{ji}} + \frac{1}{\pi_i q_{ij}}} = \pi_i q_{ij} \frac{s_{ij}}{1 + \frac{\pi_i q_{ij}}{\pi_j q_{ji}}} = \pi_i p_{ij},$$

gdzie korzystamy z symetrii s_{ij}. Zatem rodzina macierzy ergodycznych wyznaczona z rodziny $(\alpha_{ij})_{i,j \in S, i \neq j}$ (czyli w zasadzie funkcji s_{ij}) spełniających opisane warunki jest rodziną macierzy o rozkładzie stacjonarnym $\boldsymbol{\pi}$, spełniającą ponadto dla tego rozkładu *detailed balance equations*. Oznaczmy tę rodzinę przez $\mathcal{R}$. Z takiej rodziny chcemy wybrać algorytm (macierz), która posłuży nam do rozwiązania postawionych problemów. Chcemy wybrać *najlepszy* z nich – o tym, co to znaczy *najlepszy* i dlaczego jest to właśnie algorytm Metropolisa (Metropolisa-Hastingsa) opowiem w następnym podrozdziale.

3.4 Optymalność algorytmów Metropolisa i Metropolisa-Hastingsa

Opowiedzmy jeszcze raz, co chcemy robić – chcemy symulować pewien nieznany wprost rozkład prawdopodobieństwa. Symulacja jest tym lepsza, im wartości

$$\frac{1}{n} \sum_{i=0}^{n-1} f(X_i)$$

są bliższe oczekiwanych, a więc im mniejsza jest wariancja tych wartości. Okazuje się, że algorytm Metropolisa-Hastingsa jest optymalny ze względu na asymptotyczną wariancję w rodzinie $\mathcal{R}$.

Zanim to udowodnimy, zauważmy najpierw, że dla s_{ij} odpowiadających macierzom z rodziny $\mathcal{R}$ zachodzi:

$$s_{ij} \leq 1 + \min(t_{ij}, t_{ji}) = 1 + \min\left(\frac{\pi_i q_{ij}}{\pi_j q_{ji}}, \frac{\pi_j q_{ji}}{\pi_i q_{ij}}\right).$$

W przeciwnym razie zachodziłoby albo $\alpha_{ij} > 1$, albo $\alpha_{ji} > 1$, co nie może mieć miejsca, jako że każde z nich jest prawdopodobieństwem. W takim razie widać, że algorytm Metropolisa-Hastingsa maksymalizuje s_{ij}, co jest równoważne z maksymalizacją wyrazów p_{ij} leżących poza diagonalą $\mathbb{P}$.

Teraz za [9] podamy twierdzenie (bez dowodu):

Twierdzenie 3.4.1. *Niech $\mathbb{P}^{(1)} = (p_{ij}^{(1)})_{i,j \in S}$, $\mathbb{P}^{(2)} = (p_{ij}^{(2)})_{i,j \in S}$ będą dwiema ergodycznymi macierzami o stacjonarnym rozkładzie π spełniającymi ponadto detailed balance equations. Jeśli dla wszystkich wyrazów leżących poza diagonalą zachodzi $p_{ij}^{(1)} \geq p_{ij}^{(2)}$, to dla dowolnej $f : S \to \mathbb{R}$, $\mathbb{P}^{(1)}$ ma mniejszą lub równą wariancję asymptotyczną, czyli:*

$$Var_f(\mathbb{P}^{(1)}) \leq Var_f(\mathbb{P}^{(2)}).$$

Z poprzedniego rozdziału wiemy, że mówienie o wariancji asymptotycznej ma sens. Z wcześniejszych rozważań wiemy natomiast, że dla dowolnych $i \neq j \in S$ macierz z algorytmu Metropolisa/Metropolisa-Hastingsa maksymalizuje p_{ij} w obrębie rodziny $\mathcal{R}$, a więc również minimalizuje asymptotyczną wariancję w tej rodzinie. Zatem w odpowiednio długim przedziale czasowym, najlepiej symuluje docelowy rozkład. Z tego powodu ten algorytm jest najlepszym (i naszym) wyborem z tej rodziny.

Na koniec rozdziału zwróćmy uwagę, że algorytmom Metropolisa-Hastingsa i Metropolisa (dotyczy to całej omawianej rodziny) do działania wystarczy tylko możliwość obliczania *ilorazów* wartości rozkładu stacjonarnego w różnych punktach. Jest to równoważne z liczeniem ilorazów *funkcji*, do której rozkład ma być proporcjonalny. Dzięki tej pięknej własności nie musimy się przejmować już stałą normalizacyjną. Same funkcje będące bazą dla rozkładu natomiast wypływają naturalnie z opisu rzeczywistości, a obliczanie ich ilorazów nie stanowi zwykle problemu i jest *szybkie*, jeśli tylko odpowiednio skonstruuje się macierz generującą.

4 Dekodowanie zaszyfrowanego tekstu

Zajmę się w tym rozdziale zastosowaniem MCMC do dekodowania zaszyfrowanego tekstu. Opowiem zarówno o swoich *udanych* jak i pokrótce o *nieudanych* (przynajmniej do momentu oddania tej pracy) próbach rozwiązania tego problemu dla różnych typów szyfrów.

4.1 Wprowadzenie do problemu

Będę zajmował się rodzinami szyfrów podstawieniowych – mono- i polialfabetycznych (choć przede wszystkim tą drugą opcją), gdzie zaszyfrowana wiadomość jest w języku angielskim, a więc bazą najczęściej jest również angielski alfabet (bez polskich znaków). Czym są takie szyfry? Zaczniemy od przykładu jednego z najprostszych i najstarszych szyfrów – szyfru Cezara. Załóżmy, że chcemy zaszyfrować wiadomość **'MONTE CARLO MARKOV CHAINZ'**.
Mamy do dyspozycji tablicę alfabetów:

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Tabela 1: Tablica alfabetów

Szyfr Cezara składa się z jednej litery alfabetu. Niech będzie to 'B'. Zaszyfrowana wiadomość będzie miała postać **'NPOUF DBSMP NBSLPW DIBJOA'**. Szyfrogram został otrzymany w ten sposób, że każdej literze α została przyporządkowana litera na przecięciu α i litery szyfru 'B'. Można zauważyć, że jest to równoważne *przesunięciu* każdej litery tekstu o jedno miejsce w prawo (modulo długość alfabetu) – szyfry opierające się na tej zasadzie działania (przesuwaniu) nazywamy *przesuwnymi*. Szyfr Cezara jest szyfrem monoalfabetycznym, tzn. każdą literę wiadomości kodujemy przy użyciu tej samej procedury. Zanim przejdziemy do dalszych rozważań, usystematyzujmy trochę pojęcia związane z szyfrowaniem.

Definicja 4.1.1. *Alfabetem nazywamy różnowartościowy ciąg* $\boldsymbol{A} = \{\alpha_0, \alpha_1, ...\alpha_{l-1}\}$. *Element* α_i *nazywamy i-tą literą alfabetu. Wielkość l nazywamy długością alfabetu.*

Definicja 4.1.2. *(Podstawieniową) funkcją szyfrującą* σ *nad alfabetem* $\boldsymbol{A}$ *nazywamy bijekcję*

$$\sigma : \boldsymbol{A} \to \boldsymbol{A}.$$

Funkcję odwrotną σ^{-1} *nazywamy funkcją deszyfrującą.*

Zwróćmy uwagę, że $(\sigma(\alpha_0), \sigma(\alpha_1), ..., \sigma(\alpha_{l-1}))$ również zadaje alfabet. Stąd funkcję szyfrującą możemy równoważnie zdefiniować podając odpowiadający jej alfabet, czego przykład można było zobaczyć wyżej przy szyfrze Cezara (każdej możliwej literze α odpowiadał alfabet w wierszu/kolumnie α). Zwróćmy też uwagę, że symbol σ nie jest tu przypadkowy. Podstawieniowe funkcje szyfrujące są po prostu permutacjami nad alfabetem. Stąd nazwa podstawieniowa – pod każdą literę alfabetu podstawiamy inną (lub tę samą) literę alfabetu.

Definicja 4.1.3. *Tekstem* $\boldsymbol{T}$ *nad alfabetem* $\boldsymbol{A}$ *i rozszerzonym zbiorem znaków* S, $A \subseteq S$ *nazwiemy ciąg* $\boldsymbol{T} = t_0, t_1, t_2, ..., t_{n-1}$, *gdzie* $t_i \in S$. *Następnie weźmy ciąg* $\beta = \beta_0, \beta_1, ...\beta_k$ *elementów tekstu (mogą się oczywiście powtarzać), takich, że są literami alfabetu uporządkowanymi tak samo jak w oryginalnym tekście.* β_i *nazwiemy wtedy i-tą literą w tekście,* $\boldsymbol{\beta}$ *– ciągiem liter z tekstu* $\boldsymbol{T}$. *Odróżniamy ciąg liter (znaków alfabetycznych)* $\boldsymbol{\beta}$ *od ciągu wszystkich znaków.* t_i *nazwiemy i-tym znakiem tekstu* $\boldsymbol{T}$.

Zobaczmy na przykładzie, co ta definicja oznacza.

Przykład 4.1.3
Weźmy tekst 'A MARKOV CHAIN' nad alfabetem 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' i rozszerzonym zbiorem znaków składającym się z alfabetu i dodatkowo spacji. 'A' jest wtedy zerową literą β_0 (znakiem alfabetycznym) i zerowym znakiem (t_0) w ogóle. Pierwszym znakiem natomiast jest spacja (t_1), ale nie jest to pierwsza litera (ogólnie, nie jest to litera), bo spacja nie należy do alfabetu. Pierwszą literą β_1 i drugim znakiem tekstu t_2 jest 'M'.

Taka – nieco skomplikowana – definicja jest nam potrzebna, żeby można było naturalnie tekst podzielić na część szyfrowaną i nieszyfrowaną.

Definicja 4.1.4. *Kluczem szyfrującym lub kodującym nad tekstem* $\boldsymbol{T} = t_0, ..., t_{n-1}$ *o ciągu liter* $\boldsymbol{\beta} = \beta_0, ...\beta_k$ *nazwiemy ciąg (podstawieniowych) funkcji szyfrujących* $\boldsymbol{\Sigma} = \sigma_0, \sigma_1, ...\sigma_k$. *Szyfrogramem dla tekstu* $\boldsymbol{T}$ *i klucza* $\boldsymbol{\Sigma}$ *nazwiemy zaszyfrowany tekst* $\boldsymbol{E} = e_0, ..., e_{n-1}$, *gdzie* $e_j = t_j$, *jeśli* t_j *nie należy do alfabetu oraz* $e_j = \sigma_i(t_j) = \sigma_i(\beta_i)$, *jeśli* t_j *należy do alfabetu i odpowiada i-tej literze w tekście. Ciąg funkcji odwrotnych* $\sigma_0^{-1}, \sigma_1^{-1}...\sigma_k^{-1}$ *będziemy nazywać kluczem deszyfrującym bądź dekodującym.*

Innymi słowy kluczem szyfrującym (podstawieniowym) dla danego tekstu jest zbiór (podstawieniowych) funkcji szyfrujących, taki że na każdą *literę* w tekście przypada funkcja szyfrująca.

Przykład 4.1.4
Weźmy tekst 'AB BA' nad alfabetem 'AB' i rozszerzonego zbioru znaków składającego się z alfabetu i spacji. By otrzymać klucz szyfrujący, potrzebujemy tu czterech funkcji szyfrujących – po jednej na każdą literę. Niech pierwsze dwie będą identycznościowe, a następne dwie zamieniają 'A' i 'B' miejscami. Szyfrujemy: zerowy znak 'A' odpowiada zerowej literze, stosujemy więc do niego zerową funkcję szyfrującą, jest ona identycznościowa, więc zostaje 'A'. Pierwszy znak 'B' odpowiada pierwszej literze, stosujemy więc do niego pierwszą funkcję szyfrującą, jest ona identycznościowa, więc zostaje 'B'. Drugi znak ' ' (spacja) nie odpowiada literze, więc zostawiamy go bez zmian. Trzeci znak 'B' odpowiada drugiej literze, szyfrujemy go więc drugą funkcją szyfrującą, dostając 'A'. Czwarty znak 'A' odpowiada trzeciej literze, więc szyfrujemy go trzecią f. szyfrującą, dostając 'B'. Ostatecznie szyfrogram to 'AB AB'. Tabela poniżej pokazuje *odpowiadanie* dla tego tekstu: t_j odpowiada i-tej literze, gdy ma pod sobą β_i.

t_0	t_1	t_2	t_3	t_4
β_0	β_1		β_2	β_3

Zaszyfrowanie tekstu polega więc na zaszyfrowaniu każdej z liter odpowiadającą funkcją szyfrującą i pozostawieniu znaków spoza alfabetu tak jak w oryginale. Odszyfrowanie polega w takim razie na zastosowaniu klucza deszyfrującego do szyfrogramu – w wyniku otrzymamy oryginalny tekst. Może się zdarzyć, że wszystkie elementy (funkcje) *klucza* są takie same albo że co najmniej dwa spośród nich są różne. W pierwszym przypadku mówimy o kluczu (szyfrze) monoalfabetycznym (jest nim szyfr Cezara, każdą literę tekstu, szyfrujemy bowiem tą samą funkcją – odpowiadającą właściwemu alfabetowi), w drugim – o szyfrze polialfabetycznym. Możemy wyrazić teraz rodzinę Cezara funkcji szyfrujących w algebraiczny sposób, uogólniając na dowolny alfabet.

Definicja 4.1.5. *Rodziną funkcji Cezara dla alfabetu* $\boldsymbol{A} = (\alpha_0, ..., \alpha_{l-1})$ *nazwiemy funkcje postaci* $c_j(\alpha_i) = \alpha_{(i+j \mod l)}$ *dla* $j = 0, 1...l-1$.

Zauważmy prosty fakt, że rodzina funkcji (a więc i kluczy z racji monoalfabetyczności) Cezara tworzy grupę ze względu na złożenia i ta grupa jest izomorficzna z $\mathbb{Z}_l^+$ (elementem odwrotnym c_j jest $c_{(-j) \mod l}$ – jest to klucz deszyfrujący, widać więc, że klucz ten należy do tej samej rodziny). Stąd szyfr Cezara możemy utożsamiać z elementem $j \in \mathbb{Z}_l$ lub równoważnie z α_j. Odnosząc to do przedstawionego w początku rozdziału przykładu szyfr 'B' (przy alfabecie ABCDEFGHIJKLMNOPQRSTUVWXYZ) można było równoważnie zapisać jako 1, podobnie litery alfabetu można zastąpić ich pozycjami w alfabecie (A:0, B:1, ...). Dalej będziemy korzystać z obu tych zapisów zamiennie (tj. c_i będziemy zapisywać jako i lub α_i w zależności od okoliczności).

Ogólnie, klucze szyfrujące nad danym tekstem tworzą grupę ze względu na złożenia, a elementem odwrotnym jest dla danego klucza szyfrującego, odpowiadający mu klucz deszyfrujący. Algorytmy szyfrujące bazujące na kluczach podstawieniowych należą do tzw. *symetrycznych* algorytmów szyfrowania. Nazwa bierze się stąd, że znając klucz szyfrujący, jesteśmy w stanie bez problemu znaleźć klucz deszyfrujący (odwracanie permutacji jest łatwe!). Zatem do zaszyfrowania komunikacji potrzeba i wystarczy, żeby nadawca i adresat zaszyfrowanej wiadomości posiadali informację na temat tego samego klucza, zachodzi więc *symetria*. Innym rodzajem szyfrowania jest szyfrowanie *asymetryczne*, gdzie znalezienie klucza deszyfrującego mimo posiadania klucza szyfrującego jest *trudne* w pewnym sensie. W praktyce oznacza to, że w celu komunikacji nadawca zna tylko klucz szyfrujący, który może być publiczny (bo nie można łatwo z niego odtworzyć klucza deszyfrującego), natomiast odbiorca zna klucz deszyfrujący – ten dla bezpieczeństwa komunikacji powinien być tajny. Ze względu na asymetrię: tajny-publiczny szyfrowanie to nazywamy asymetrycznym. Ataki na szyfry asymetryczne to bardzo ciekawy temat, jednak nie będę się nim zajmował w tej pracy. Przykładami szyfrów asymetrycznych są ECC czy RSA.

Celem tej części pracy będą *ataki* na różne szyfry podstawieniowe (przede wszystkim pewne szczególne przypadki szyfrów polialfabetycznych). Atak na szyfr to próba zdekodowania zaszyfrowanej tym szyfrem wiadomości (szyfrogramu), mimo braku znajomości klucza deszyfrującego – czyli innymi słowy próba *zgadnięcia* tego klucza.

Wróćmy do szyfru Cezara – nazwa pochodzi stąd, że według podań historycznych Juliusz Cezar używał tego szyfru do kodowania swoich wiadomości do przyjaciół [10] – i naszej zaszyfrowanej wiadomości: **NPOUF DBSMP NBSLPW DIBJOA** Zwróćmy uwagę, że szyfr Cezara jest bardzo łatwy do złamania komputerowo za pomocą ataku *brute force* – wystarczy sprawdzić wszystkie l (długość alfabetu) możliwości klucza deszyfrującego – w naszym wypadku 26 i – jeśli tylko tekst nie jest za krótki (zauważmy bowiem, że możliwymi dekodowaniami szyfrogramu ST są zarówno NO jak i HI), powinien być tylko jeden klucz deszyfrujący, dla którego odkodowana wiadomość ma sens.

0: NPOUF DBSMP NBSLPW DIBJOA
1: OQPVG ECTNQ OCTMQX EJCKPB
...
23: PRQWH FDUOR PDUNRY FKDLQC
24: LNMSD BZQKN LZQJNU BGZHMY
25: MONTE CARLO MARKOV CHAINZ

Istotnie, tylko dla deszyfrującego klucza 25 (równoważnie: Z lub -1), wiadomość ma sens, więc odgadujemy, że to jest właśnie klucz deszyfrujący,a MONTE CARLO MARKOV CHAINZ jest zaszyfrowaną wiadomością. Ważne jest założenie, że oryginalna wiadomość powinna mieć pewne cechy znanego języka. Jeśli bowiem wiadomość byłaby nadana w jakimś dziwnym, nieznanym nam języku, w którym OQPVG ECTNQ OCTMQX EJCKPB ma znaczenie, a MONTE CARLO MARKOV CHAINZ – nie, atak nie powiedzie się (o ile nie znajdziemy do pomocy kogoś, kto posługuje się tym dziwnym językiem). Tak więc metoda *brute force* opiera się na znajomości pewnych struktur w rzeczywistym języku – w tym wypadku angielskim – przyjmujemy założenie, że oryginalna wiadomość została napisana właśnie w tym rzeczywistym języku. Na tych samych założeniach będą się opierały wszystkie dalej opisane metody.
Jeśli nie mielibyśmy do dyspozycji komputera, wypisywanie wszystkich możliwości byłoby czasochłonne, atak da się ulepszyć wtedy przy użyciu analizy częstotliwościowej: tj. wiadomo, że najczęściej występującymi literami w języku angielskim są E,T,A,O – możemy zatem założyć, że najczęściej występującą literę szyfrogramu klucz deszyfrujący powinien przeprowadzić na jedną z nich – oczywiście wyznaczenie takiego klucza jest bardzo łatwe. Szybkie sprawdzenie tych czterech możliwości powinno nam dać właściwą.

Przejdźmy teraz do następnego rozdziału, w którym opiszę, jakimi będę się zajmował szyframi. Większość z nich opiera się mniej lub bardziej na idei szyfru *przesuwnego* (Cezara), są jednak szyframi polialfabetycznymi, co czyni ich złamanie znacząco trudniejszym.

4.2 Przedstawienie analizowanych szyfrów

Chcemy użyć idei szyfru przesuwnego. Szyfr Cezara jest zbyt prosty, ale umiemy również w prosty sposób go skomplikować. Wystarczy zestawić kilka (k) szyfrów Cezara w następujący sposób: wróćmy do naszej wcześniejszej wiadomości: MONTE CARLO MARKOV CHAINZ (w zapisie liczbowym $\mathbb{Z}_{26}$:

$$[12, 14, 13, 19, 4,'', 2, 0, 17, 11, 14,'', 12, 0, 17, 10, 14, 21,'', 2, 7, 0, 8, 13, 25]$$

Wcześniej szyfrowaliśmy ją za pomocą klucza 'B' (odpowiadającego 1 w $\mathbb{Z}_{26}^{+}$). Rozważmy teraz klucz 'BC' (czyli $[1, 2]$). Szyfrowanie przeprowadzamy w następujący sposób:

M	O	N	T	E		C	A	R	L	O		M	A	R	K	O	V		C	H	A	I	N	Z
B	C	B	C	B		C	B	C	B	C		B	C	B	C	B	C		B	C	B	C	B	C	

Tj. dla zerowej litery (numerujemy od zera, bo jest to wygodniejsze z punktu widzenia dodawania indeksów *modulo*), czyli M, używamy klucza B (inaczej 1), dostając N, dla pierwszej litery, czyli O, używamy klucza C (2) i dostajemy Q itd., spacje jako znaki spoza alfabetu zostawiamy bez zmian. Ostatecznie szyfrogram ma postać:

NQOVF EBTMQ NCSMPX DJBKOB

Co odpowiada zapisowi liczbowemu:

$$[13, 16, 14, 21, 5,'', 4, 1, 19, 12, 16,'', 13, 2, 18, 12, 15, 23,'', 3, 9, 1, 10, 14, 1]$$

Ogólnie, jak widać, kopiujemy klucz, 'tyle razy ile potrzeba', aby móc zaszyfrować cały tekst – tak samo robiliśmy przy szyfrze Cezara, z tym że klucz tam był jednoelementowy (tu jest 2-elementowy, ogólnie może być n-elementowy).

Możemy zatem wprowadzić rodzinę szyfrów: szyfry z tej rodziny nazywamy szyframi Vigenère'a od nazwiska Blaise'a de Vigenère'a XVI-wiecznego francuskiego kryptografa, któremu błędnie przypisywany jest pomysł tego szyfru [11]. Szyfr Vigenère'a w rzeczywistości pierwszy raz został opisany przez włoskiego kryptografa Giovana Battistę Bellaso. Sam Vigenère jest natomiast autorem szyfru z autokluczem [12] opierającego się na podobnej idei – tym szyfrem również zajmiemy się później.

Definicja 4.2.1. *Szyframi Vigènere'a długości n nad tekstem* $\boldsymbol{T}$ *o ciągu liter* $\beta_0, ..., \beta_k$ *nazywamy szyfry postaci* $\boldsymbol{\Sigma} = \sigma_0, \sigma_1, ..., \sigma_k$ *wyznaczane przez ciąg funkcji szyfrujących Cezara* $i_0, ..., i_{n-1}$ *w następujący sposób* $\sigma_m = i_{(m \mod n)}$.

Łatwo zauważyć, że już dla nie tak wielkich n zastosowanie metody *brute force* nie jest możliwe, liczba możliwych szyfrów w rodzinie wynosi wtedy l^n, gdzie l jest długością alfabetu, a więc rośnie wykładniczo wraz z długością szyfru. Zauważmy, że szyfry Vigenère'a definiują się przez ciąg funkcji Cezara, a więc tak naprawdę przez ciąg elementów $\mathbb{Z}_l$. Łatwo zauważyć, że klucz deszyfrujący Vigenère'a bądź złożenie kluczy o długości n również będą zdefiniowane przez n szyfrów Cezara: dokładnie $-i_0, -i_1, ..., -i_{n-1}$ (skopiowanych tak, by zostały pokryte wszystkie litery w tekście) w konwencji $\mathbb{Z}_l^+$ dla klucza szyfrującego $i_0, ..., i_{n-1}$, a więc należy również do tej samej rodziny szyfrów oraz przy znajomości $i_0, ..., i_{n-1}$ nie zależy od kodowanego/dekodowanego tekstu. Jest to o tyle wygodne, że do celów kodowania i dekodowania można wówczas użyć tej samej funkcji zależnej tylko od $i_0, ..., i_{n-1}$ – rodzina tych szyfrów jest izomorficzna z $\mathbb{Z}_l^n(+)$, i nie zależy od szyfrowanego tekstu. W kolejnym przypadku – szyfru z autokluczem – sprawa będzie się miała inaczej.

Opiszmy teraz szyfr z autokluczem, ten, którego autorem istotnie jest Blaise de Vigenère. Zaczniemy od przykładu, znów chcemy zakodować wiadomość MONTE CARLO MARKOV CHAINZ. Szyfr Vigenère'a z autokluczem wyraża się podobnie jak zwykły szyfr Vigenère'a przez ciąg liter lub elementów $\mathbb{Z}_l$. Użyjmy znów klucza szyfrującego $(1, 2)$ (BC) i sporządźmy tabelę szyfrowania:

M	O	N	T	E	C	A	R	L	O	M	A	R	K	O	V	C	H	A	I	N	Z
B	C	M	O	N	T	E	C	A	R	L	O	M	A	R	K	O	V	C	H	A	I

Gdzie szyfr możemy zapisać również jako:

$$[1, 2, 12, 14, 13, 19, 4, 2, 0, 17, 11, 14, 12, 0, 17, 10, 14, 21, 2, 7, 0, 8]$$

Widać jasno ideę szyfru: dla szyfru długości n pierwsze n liter koduje się za pomocą szyfru, tak jak w przypadku zwykłego szyfru Vigenère'a, a następnie do zaszyfrowania używa się samej wiadomości – stąd nazwa szyfru z autokluczem (*autokey*), wiadomość jest bowiem sama częścią klucza. Otrzymujemy szyfrogram:

NQZHR VETLF XODKFF QCCPNH.

Możemy sformalizować definicję szyfru (Vigenère'a) z autokluczem.

Definicja 4.2.2. *Szyframi (Vigènere'a) z autokluczem długości n nad tekstem $\boldsymbol{T}$ o ciągu liter $\beta_0, ..., \beta_k$ nazywamy szyfry $\boldsymbol{\Sigma} = \sigma_0, \sigma_1, ..., \sigma_k$ wyznaczane przez ciągi funkcji szyfrujących Cezara $i_0, ..., i_{n-1}$ w następujący sposób:*

$$\sigma_m = \begin{cases} i_m \text{ jeśli } m < n \\ \beta_{m-n} \text{ w przeciwnym wypadku,} \end{cases}$$

gdzie β_{m-n} utożsamiamy z odpowiadającą tej literze funkcją Cezara (podobnie będziemy utożsamiać litery i funkcje Cezara z elementami grupy addytywnej $\mathbb{Z}_l$ wszędzie później, np. dla klasycznego alfabetu -A=A, bo $-0 \equiv 0 \mod 26$, natomiast -B=Z, bo $-1 \equiv 25 \mod 26$).

Istotną kwestią jest teraz to, że do dekodowania szyfrogramu (który powstał przy użyciu klucza kodującego wygenerowanego przy pomocy ciągu Cezara $i_0, ..., i_{n-1}$) o ciągu liter $\gamma_0, ...\gamma_k$ należy użyć klucza $\delta_0, ..., \delta_k$ postaci:

$$\delta_m = \begin{cases} -i_m \text{ jeśli } m < n \\ -\delta_{m-n}(\gamma_{m-n}) \text{ w przeciwnym wypadku.} \end{cases}$$

Dla udowodnienia poprawności klucza można przeprowadzić prostą indukcję: oczywiście dla $m < n$ wiadomość została zdekodowana poprawnie. Dla większych m zakładamy, że wszystkie wcześniejsze litery zostały zdekodowane poprawnie. γ_m została zakodowana przez β_{m-n} zgodnie z definicją autoklucza. Zatem zostanie odkodowana przez przesunięcie $-\beta_{m-n}$. Ale $\beta_{m-n} = \delta_{m-n}(\gamma_{m-n})$, ponieważ założyliśmy, że wcześniejsze litery zostały zdekodowane poprawnie, stąd δ jest poprawnym kluczem dekodującym.

W tym wypadku jak widać, nie można kodowania i dekodowania zaimplementować tą samą funkcją (jeśli za argumenty funkcji dekodującej mamy do dyspozycji tylko $-i_0, ... - i_{n-1}$ i szyfrogram). Jednak algorytm deszyfrujący nie jest istotnie bardziej skomplikowany niż algorytm kodujący – do dekodowania kolejnych litery używamy po prostu liter dotychczas zdekodowanych.

Na koniec zauważmy, że możemy jeszcze nieco bardziej skorzystać z utożsamienia alfabetu z $\mathbb{Z}_l$. Mianowicie stosując ideę taką jak dla szyfru Vigenère'a (czyli kopiowanie n-elementowych ciągów funkcji szyfrujących 'tyle razy, ile trzeba') szyfrować i nie przez $i + b \mod l$ dla $b \in \mathbb{Z}_l$ jak dla szyfru Vigenère'a, ale przez $ai + b \mod l$, gdzie $a \in \mathbb{Z}_l^*$ i $b \in \mathbb{Z}_l$ – element klucza odpowiadający danej współrzędnej możemy zatem zapisać jako (a, b).

Definicja 4.2.3. *Szyfry z rodziny opisanej powyżej będę nazywać rozszerzonymi/afinicznymi szyframi Vigenère'a.*

Nazwa *rozszerzony/afiniczny* jest wprowadzona przeze mnie, szyfr ten nie ma powszechnie przyjętej osobnej nazwy, jest traktowany po prostu jako jedna z wariacji szyfru Vigenère'a. Klucz dekodujący dla rozszerzonego klucza kodującego na poszczególnych współrzędnych będzie miał postać $(a^{-1}, -a^{-1}b)$, bowiem $a^{-1}(ai + b) - a^{-1}b \equiv i \mod l$, gdzie a^{-1} jest odwrotnym elementem do a w grupie $\mathbb{Z}_l^*$ z mnożeniem $\cdot_l$ Stąd do kodowania i dekodowania możemy użyć tej samej implementacji niezależnej od kodowanego/dekodowanego tekstu, podobnie złożenie należy do tej samej rodziny kluczy niezależnych od tekstu – szyfry te tworzą grupę niezależną od szyfrowanego tekstu izomorficzną z iloczynem $(\mathbb{Z}_l^*, \mathbb{Z}_l^+)^n$. Dla przykładu zobaczmy napis MONTE CARLO MARKOV CHAINZ zaszyfrowany za pomocą klucza $[(1, 1), (3, 2)]$:

NSOHF IBBMS NCSGPN DXBAOZ

Wyliczamy pierwszą literę szyfrogramu: $1 * M + 1 \equiv 1 * 12 + 1 \equiv 13 \mod 26 \equiv N$, drugą: $3 * O + 2 \equiv 3 * 14 + 2 \equiv 20 \mod 26 \equiv S$ itd.

Poza wymienionymi tu szyframi zajmę się jeszcze w dalszej analizie szyfrem podstawieniowym (raczej pokrótce: podejście MCMC do tego szyfru zostało już obszernie przeanalizowane w [1], [2], [3], mi niestety nie udało się znacząco rozszerzyć tamtych wyników, więc opiszę tylko skrótowo swoje próby i spostrzeżenia), tj. wspomnianym już szyfrem monoalfabetycznym: szyfrowanie polega na zastosowaniu do każdej litery tekstu tej samej permutacji alfabetu.

4.3 Idea ataku na przedstawione szyfry

Tak jak już wspomniałem, do ataku na przedstawione szyfry wykorzystam pewne własności języka angielskiego – mianowicie częstości występowania n-gramów w tym języku. n-gramem dla zbioru znaków S nazwiemy ciąg $g \in S^n$. Przykładowo 4-gramem ze zbioru znaków $\{ABCD\_\}$ będzie np. $ADC\_$. Ze względu na znikomy (albo wręcz negatywny) wpływ na skuteczność ataku, natomiast na pewno negatywny wpływ na efektywność zwiększania n (rozmiar słownika n-gramów rośnie wykładniczo ze względu na n, więc im większe n tym większa złożoność pamięciowa) odnotowany już z samego początku pracy ograniczyłem się do analizy 1-gramów (inaczej *monogramów* lub *unigramów*: ta pierwsza nazwa prawdopodobnie nie jest do końca poprawna, jako że monogram ma swoje inne znaczenie w języku polskim, jednak z początku pracowałem na słownikach częstości z [30], gdzie nazwa ta się pojawia – z tego powodu powszechnie używałem jej w kodzie, więc również tu powinna się pojawić dla osiągnięcia pewnej spójności), 2-gramów (inaczej *bigramów* lub *digramów*) oraz 3-gramów (inaczej *trigramów).*

Stosujemy piękne podejście znane już wcześniej z prac [1], [2], [3].
i-tym n-gramem tekstu $\boldsymbol{T} = t_0...t_{|T|-1}$ nazwiemy $t_i t_{i+1}...t_{i+n-1}$ i będziemy go oznaczać po prostu przez $t_i(n)$. Załóżmy, że mamy dane częstości (niekoniecznie znormalizowane) występowania n-gramów $f_n(g)$ w języku, w którym była napisana zaszyfrowana w wiadomość (zakładamy, że wiemy, w jakim, i że go znamy) i szyfrogram $\boldsymbol{E} = e_0, ...e_{|T|-1}$, zaszyfrowany tekst, który chcemy odszyfrować. Dla potencjalnego klucza dekodującego $\boldsymbol{k}$, który zadaje tekst – zdekodowany szyfrogram

$D = d_0^{\boldsymbol{k}}, ..., d_n^{\boldsymbol{k}}$ wprowadzamy w_n – *wagę ze względu na n-gram* takiego klucza:

$$w_n(\boldsymbol{k}) = \prod_{i=0}^{|T|-n} f_n(d_i^{\boldsymbol{k}}(n))$$

Można wprowadzić *wagę mieszaną* ze współczynnikami p_n, tj.

$$w(\boldsymbol{k}) = \prod_{n=1}^{\infty} (w_n(\boldsymbol{k}))^{p_n}$$

(oczywiście współczynniki powinny być niezerowe tylko dla skończenie wielu n). W praktyce – choć przeprowadzałem początkowo pewne doświadczenia z powyższą mieszaną wagą, jej zastosowanie nie poprawiało w żaden sposób jakości ataku (choć można przeprowadzić dalsze badania na ten temat, bo prawdopodobnie nie został on wyczerpany). Ostatecznie więc używałem wszędzie 'czystych' n-gramowych wag dla różnych n – podobnie użycie współczynnika p_n różnego od 1 dla n-gramowych wag nie przyniosło istotnej poprawy, więc dalej będą stosowane po prostu czyste n-gramowe wagi.

Dalej możemy wprowadzić równoważną definicję n-gramowej wagi. Dla wszystkich możliwych n-gramów g nad użytym zbiorem znaków S – niech zadają one zbiór G – dla potencjalnego dekodowania szyfrogramu D oznaczmy przez $r_D(g)$ liczbę wystąpień g w D. Wtedy:

$$w_n(\boldsymbol{k}) = \prod_{g \in G} f_n(g)^{r_D(g)}$$

Intuicyjnie, waga jest tym większ,a im bardziej tekst jest bliski językowi angielskiemu – a więc rzeczywistej wiadomości – bowiem wtedy popularne n-gramy występują często, a te niepopularne rzadko. Użyliśmy tu wagi jak w [3]. Można użyć również ciut innego podejścia [2], [1], które – mimo że mniej wygodne – ma ładne Markowowskie uzasadnienie.

Dla unigramów funkcja stanu/waga w tym podejściu będzie taka sama: uznaje się wtedy, że język angielski generuje kolejne znaki niezależnie zgodnie z rozkładem f_1 (znormalizowanym). Dla większych n przyjmuje się, że język angielski generuje łańcuch Markowa o rozkładzie początkowym f_1 i macierzy przejścia wyliczonej z f (w sposób związany z n-gramami). Podam przykładowo koncepcję dla bigramów. Pierwszy (zerowy) znak s_0 generujemy z rozkładu f_1. Następne znaki generujemy, biorąc pod uwagę jeden poprzedni, zgodnie z macierzą przejścia:

$$p_{ij} = P(s_{n+1} = j | s_n = i) \approx \frac{f_2(ij)}{f_1(i)}$$

Ostatnia równość bierze się stąd, że $\sum_j f_2(ij) \approx f_1(i)$, występuje znak $\approx$ zamiast $=$ ponieważ po lewej stronie nie bierzemy pod uwagę sytuacji, gdzie i występuje na końcu tekstu – co można jednak łatwo naprawić, dodając znak końca tekstu do naszego zbioru znaków. Wtedy funkcja wagi miałaby postać:

$$f(d_0) \cdot \prod_{i=1}^{|T|-2} p_{d_i d_{i+1}}$$

Użycie takiej wagi ma bardzo ładne teoretyczne uzasadnienie, ale z przeprowadzonych doświadczeń użycie jej nie zwiększa mocy ataku w porównaniu z wagą zdefiniowaną wcześniej. Są natomiast pewne małe implementacyjne niedogodności – np. specjalnie trzeba traktować pierwszy znak tekstu, trzeba normalizować częstości – więc w swoich atakach użyłem zdefiniowanej wcześniej wersji zgodnej z [3].

Co zatem będzie ideą naszego ataku – można się domyślić, poszukiwanie:

$$\arg\max_{\boldsymbol{k}} w_n(\boldsymbol{k})$$

dla różnych n. Jeśli znajdziemy taki klucz, który maksymalizuje n-gramową wagę, odszyfrowany tekst powinien być blisko języka angielskiego, a skoro tak, to jest to prawdopodobnie poprawnie

odszyfrowana wiadomość.
Koncepcja tu zaprezentowana słusznie może kojarzyć się z wnioskowaniem Bayesowskim, bo jest jego wersją. Rozkładem *a priori* jest tutaj jednostajny rozkład na zbiorze kluczy z danej rodziny (nic nam nie wiadomo o tym, żeby np. przesunięcia o parzystą liczbę były stosowane częściej). Doświadczeniem czy obserwacją jest dany nam szyfrogram. Nieznanym parametrem jest klucz $\boldsymbol{k}$, natomiast waga – po znormalizowaniu – jest naszym rozkładem *a posteriori*.
W następnym podrozdziale opowiem, jak poszukiwać klucza maksymalizującego prawdopodobieństwo *a posteriori*, co będzie rozwiązaniem problemu dekodowania.

Istotna uwaga: ze względów numerycznych dużo lepszą opcją jest badanie

$$\log w_n(\boldsymbol{k}) = \sum_{i=0}^{|T|-n} \log f_n(d_i^{\boldsymbol{k}}(n)) = \sum_{g \in G} r_D(g) f_n(g).$$

Oczywiście jako że logarytm jest funkcją rosnącą, to arg max będzie w obu przypadkach ten sam, więc są to opcje równoważne. Toteż lepiej jest przechowywać słownik logarytmów częstości – (1) dodawanie jest tańsze, (2) operujemy na mniejszych wartościach, więc są też mniejsze błędy związane z arytmetyką zmiennoprzecinkową i nie ma ryzyka przekroczenia maksymalnej/minimalnej reprezentowalnej przez liczby zmiennoprzecinkowe wartości.

Ostatecznie więc zakładamy, że przechowujemy taki (wyliczony ze zbioru uczącego bądź w jakiś sposób dany z góry) słownik log-częstości (dla n-gramów) i mamy do niego dostęp w czasie stałym.
Nasze słowniki wyliczaliśmy na podstawie trzech pozycji (w angielskiej wersji) z wolnej internetowej biblioteki *Gutenberg project* [31]: *Wojny i pokoju* Lwa Tołstoja, *Olivera Twista* Karola Dickensa oraz *Dumy i uprzedzenia* Jane Austen.

4.4 Algorytmy deterministyczne

4.4.1 Wyprowadzenie i opis

Zaczynamy więc część właściwą pracy. Swoje rozwiązania będę prezentował na razie dla zwykłego szyfru Vigenère'a (uogólnienie ich na pozostałe szyfry (autoklucz, szyfr afiniczny) wymaga jedynie doprecyzowania kroków występujących w podprocedurach) na alfabecie $\boldsymbol{A}$ o długości l – na razie przy założeniu znajomości długości klucza $\boldsymbol{k}$ – oznaczmy ją przez n_k. Mamy dany tekst–szyfrogram o długości m: $\boldsymbol{E} = e_0 e_1 ... e_{m-1}$. Zacznijmy od najprostszej kwestii – maksymalizacji wagi unigramowej (jej logarytmu). Wróćmy do napisu MONTE CARLO MARKOV CHAINZ i szyfru $[1, 2]$. Dostajemy zaszyfrowaną wiadomość: NQOVF DCSNP NCSMPX EICJPA. Zauważmy teraz, że szyfrogram możemy podzielić na 2 fragmenty zaszyfrowane monoalfabetycznym szyfrem Cezara, mianowicie na *litery* z pozycji (liczymy tylko pozycje liter – elementów szyfrowanych) $\equiv 0 \mod 2$:

NOFCNNSPECP

oraz $\equiv 1 \mod 2$:

QVDSPCMXIJA

Ogólnie, dla szyfru długości n_k możemy podzielić napis na n_k monoalfabetycznych ciągów liter (pozycje $\equiv 0, 1, 2, 3, ...n_k - 1 \mod n_k$. Dla każdego z nich mamy l możliwych kluczy dekodujących do sprawdzenia, obliczenie *cząstkowej* (tj. związanej z danym ciągiem uzyskanym przez podział) logarytmu wagi każdej możliwości kosztuje nas m/n_k dodawań, możliwości jest l, monoalfabetycznych ciągów n_k. Oznaczmy wagę cząstkową klucza k_i o długości 1 użytego do dekodowania na pozycjach $\equiv i \mod l$ przez $w_1^{(i)}(k_i)$ Ostatecznie więc rozwiązanie wygląda tak:

Algorytm 3: Algorytm monogramowy

```
1 klucz_dekodujący = []
  dla: i = 0, 1, 2, 3...n_k − 1
2    klucz_dekodujący[i] = arg max_{k_i=0,1...l−1} log w_1^(i)(k_i);
  zwróć: klucz_dekodujący
```

a zgodnie z wcześniejszymi rozważaniami złożoność czasowa rozwiązania to $O(m/n_k \cdot n_k \cdot l) = O(ml)$. Rozwiązanie jest więc proste i szybkie, jednak działa dobrze tylko dla kluczy krótszych od pewnej wielkości zależnej od długości tekstu m, co pokażemy doświadczalnie później. Można jedynie zauważyć, że ma to związek z tym, że metoda unigramowa nie bierze w ogóle pod uwagę związków między kolejnymi znakami tekstu. Te związki w oczywisty sposób są już brane pod uwagę dla n-gramów o większych n, co daje im sporą przewagę. Wynik, który otrzymaliśmy tym podejściem (przy użyciu wygenerowanego ze zbioru uczącego: *Wojna i pokój*, *Oliver Twist*, *Duma i uprzedzenie*, słownika) dla naszego szyfrogramu to: MANFE OADLA MMRWOH CTAUNL. Widać więc, że trafiliśmy w 50%, konkretnie na ciągu związanym z $0 \mod 2$, na drugim ciągu, jak widać, zaszło odchylenie od oczekiwanych wartości, na tyle, że nasza metoda zawiodła.

Zaprezentuję teraz jak rozwiązywać w sposób deterministyczny postawione zagadnienie dla $n > 1$, konkretnie pokażę jak zrobić to dla $n = 2$ – idea potem rozszerza się na większe n w naturalny – choć niewygodny z punktu widzenia implementacji – sposób.
Wracamy do naszego tekstu MONTE CARLO MARKOV CHAINZ, szyfrogramu zakodowanego, tym razem 3-elementowym kluczem $[1, 2, 3]$, aby lepiej pokazać, o co chodzi w metodzie. Szyfrogram ma postać: NQQUG FBTOP ODSMRW EKBKQA. Stosujemy podobny trick, co wcześniej – szyfrogram można podzielić na 3 (ogólnie n_k) ciągi bigramów, zaszyfrowanych tym samym 2-elementowym kluczem (kolejno na pozycjach (0,1), (1,2) i (2,0):
dla (0,1):
NQ, UG, BT, para bigramów: P+spacja i spacja+O (tak że P jest wynikiem zaszyfrowania 0-owym elementem klucza, a O 1-szym), **SM, W+spacja, spacja+E. BK**
Pewnym usprawnieniem, które wprowadziłem, jest, jak widać, branie pod uwagę w analizie statystycznej również połączeń między szyfrowanymi (należącymi do alfabetu) a nieszyfrowanymi (niealfabetycznymi) znakami (tak jak w poprzednim podrozdziale było napisane: bierzemy n-gramy nad całym zbiorem znaków, które występują w zbiorze uczącym), co pozwala uchwycić więcej zależności statystycznych. Zauważmy, że przy zmianie szyfru zmieniają się tylko znaki alfabetyczne, stąd np. do jakichkolwiek obliczeń związanych z szyfrowaniem wystarczy brać pod uwagę tylko te n-gramy, które mają w sobie choć jeden alfabetyczny znak. Podobnie bierzemy bigramy związane z (1,2) i (2,0). Znowu przyda nam się *waga cząstkowa* ($w_2^{(i,i+1)}(k_i, k_{i+1})$ związana z ciągiem elementów na pozycjach $\equiv i, i+1 \mod n_k$ dla $i = 0, 1..., n_k - 1$. Dla $i = n_k - 1$ oczywiście sumujemy tylko log-wagi bigramów $(n_k - 1, n_k)$, $(2n_k - 1, 2n_k)$ itd. – które próbujemy odkodować 'podkluczem' k_{n_k-1}, k_0). Nie możemy podejść do tego problemu jak poprzednio i po prostu policzyć dla każdego ciągu bigramów l^2 możliwości i wybrać z nich argument maksymalizujący (2-elementowy klucz dekodujący), bo może się zdarzyć (i zwykle się zdarza!), że dla pozycji (0,1) uzyskamy np. [1,7] a dla pozycji (1,2), [19, 3] – innymi słowy: po prostu nie zgadzają się wartości – w tym wypadku na współrzędnej 1. Jesteśmy jednak sobie w stanie z tym poradzić, choć nie za darmo – bo w zamian tracimy na złożoności czasowej i pamięciowej.

Pomysł opiera się na programowaniu dynamicznym (warto zwrócić uwagę na podobieństwo algorytmu do algorytmu Viterbiego [23], co ładnie nawiązuje do łańcuchów Markowa). Ideą jest przechowywać *maksymalny prefiks o długości d+1, zerowej pozycji i oraz d-tej pozycji j*. Dla wszystkich $i, j = 0, 1...l - 1$ oraz d rosnących od 1 do n_k. Co znaczą te maksymalizujące prefiksy? Są to takie prefiksy klucza (np. prefiksami [1,2,3] są [1], [1,2], [1,2,3]), że wartość log-wagi bigramowej ograniczona do pozycji $[0, 1, 2, ...d - 1]$ jest maksymalna (umiemy to kolejno obliczać). Po ostatnim kroku mamy więc maksymalne wartości prefiksów (i możliwość odtworzenia ich całości, jak to w programowaniu dynamicznym) o długości $n_k + 1$ dla wszystkich możliwych par początek-koniec prefiksu. Jednak zwróćmy uwagę, że jedyne pary, które mają sens to takie, gdzie koniec jest równy początkowi (bo szyfr ma długość n_k, więc element szyfru na pozycji 0 musi się równać elementowi na pozycji n_k). Wybieramy więc spośród tych par tę, która maksymalizuje wagę a następnie odtwarzamy szyfr. Istnienie takiego algorytmu jest o tyle istotne, że daje nam narzędzia, aby w skończonym czasie ocenić skuteczność metody bigramów (podejście MCMC, o którym napiszę później, nie jest deterministyczne, stąd jego wyniki mogą być obarczone np. niedoskonałością generatora pseudolosowego), w tym sensie, że możemy stwierdzić, do kiedy $\arg\max$ rzeczywiście pokrywa się z kluczem dekodującym. Jako że jest to główny wynik tego podrozdziału, poświęcę mu chwilę więcej uwagi i opisowo udowodnię też jego poprawność.
Zaczniemy od opisania podprocedury przechodzenia z d-tego do $d + 1$-szego kroku, na razie pomińmy część odzyskiwania klucza (będzie to później proste ulepszenie) i skupmy się na znajdowaniu

maksymalnej wagi (logarytmu) i elementu $i \in \mathbb{Z}_l$, który stoi na początku *najlepszego* klucza. Załóżmy, że mamy do dyspozycji tabelę $T_d[i][j]$ dla wszystkich $i, j \in \mathbb{Z}_l$, która w aktualnym momencie ma w sobie maksymalną wagę log-wagę, którą da się osiągnąć dekodując tekst kluczem o zerowym elemencie i oraz d-tym elemencie j, wliczając do wagi tylko bigramy na pozycjach od 0 do d. Na tej podstawie, chcemy obliczyć maksymalną log-wagę dla i, j i $d+1$, którą następnie umieścimy w nowej tabeli $T_{d+1}[i][j]$ (poprzednią tabelę musimy pozostawić niezmienioną, dopóki krok d się nie skończy). Bierzemy więc:

$$T_1[i][j] = w_2^{(0,1)}(i,j)$$
$$T_{d+1}[i][j] = \max_{k \in \mathbb{Z}_l}(T_d[i,k] + w_2^{(d,d+1)}(k,j)), \;\; d > 1.$$

Z argumentu indukcyjnego jasne jest, że dla wszystkich kroków (także do ostatniego odnosi się ten argument, choć zajmiemy się nim osobno, bo jest szczególny) będziemy w nowej tabeli przechowywać wartości zgodne z założeniami, czyli maksymalną wartość wagi ograniczonej do pozycji $0, 1...d+1$, wystarczy zauważyć, że przy ustalonym k z założenia indukcyjnego dostajemy maksymalną wartość log-wagi dla prefiksów takich, że na zerowym miejscu jest i, na d jest k, a na $d+1$ jest j. Ponieważ iterujemy po wszystkich k, więc dostajemy maksimum globalne. Jest to standardowa procedura w algorytmach dynamicznych – parametr k współdzielony przez obydwa składniki sumy zapewnia nam spójność klucza, który w ten sposób uzyskujemy. Pewne wątpliwości może budzić ostatni krok, który robi swego rodzaju 'zawinięcie' (przeprowadzić go wystarczy, tylko dla par i, i, jak wspomniałem wcześniej), czyli:

$$T_{n_k}[i][i] = \max_{k \in \mathbb{Z}_l}(T_{n_k-1}[i,k] + w_2^{(n_k-1,n_k)}(k,i)).$$

Ale w rzeczywistości on też jest poprawny z tego samego argumentu, bowiem przy ustalonym k pierwsza część sumy maksymalizuje wagę cząstkową *złożenia* bigramów $(0,1)$, $(1,2)$, ... $(n_k - 2, n_k - 1)$ z założenia indukcyjnego, a druga część jest jedyną możliwą do wyboru opcją. Iterując po wszystkich k dostaniemy maksimum globalne. Maksimum z $T_{n_k}[i][i]$ po $i \in \mathbb{Z}_l$ da więc rzeczywiście maksymalną log-wagę bigramową, algorytm jest więc poprawny.

Zanim zapiszemy algorytm w spójnej wersji – parę uwag. Ponieważ przechowujemy pełen szyfrogram, łącznie z niezakodowanymi (niealfabetycznymi) znakami (co więcej, bierzemy je też pod uwagę przy obliczeniach), a z drugiej strony, chcemy móc szybko iterować po znakach zakodowanych (tzn. przy obliczaniu cząstkowej log-wagi chcemy poruszać się skokami o n_k – żeby dekodować kolejne pozycje równe $\mod n_k$) – mieć po prostu stały dostęp (O(1)) do i-tego elementu. Nie jest to jednak bardzo trudne – wystarczy użyć przeznaczonej do tego struktury, która:
(a) będzie w ciągłej pamięci przechowywać zakodowane znaki nieprzedzielone niezakodowanymi,
(b) będzie dawała możliwość sprawdzenia w stałym czasie, czy po danej pozycji nie zaczyna się ciąg znaków niezakodowanych: jeśli tak, będzie przekazane coś w rodzaju wskaźnika do tego ciągu znaków (w przypadku Pythona może to być po prostu indeks na liście ciągów niezakodowanych znaków), który również przechowujemy w pamięci ciągłej, żeby np. mieć szybki dostęp do pierwszego i ostatniego znaku w tym ciągu.

Spełnienie (a) i (b) zapewnia nam, że wszystkie operacje związane z obliczaniami n-gramowymi odbędą się w najkrótszym możliwym czasie.

Wróćmy teraz do algorytmu bigramowego. Jak odzyskać maksymalizujący klucz? Wystarczy przechowywać jeszcze jedną tablicę $P[i][j][d]$, która zawiera poprzedników j w maksymalizującym prefiksie o początku i, i końcu j na pozycji d. Czyli:

$$P[i][j][1] = i$$
$$P[i][j][d+1] = \arg\max_{k \in \mathbb{Z}_l}(T_d[i,k] + w_2^{(d,d+1)}(k,j)), \;\; d > 1.$$

Podobny argument jak poprzednio pozwala nam stwierdzić, że rzeczywiście po każdym kroku elementy przechowywane w $P[i][j][d+1]$ i wcześniejszych tablicach są zgodne z założeniem. Pozostaje odzyskać klucz z tablicy P. Załóżmy, że j jest elementem, który występuje na zerowej pozycji maksymalizującego klucza. Możemy wtedy podać procedurę na odzyskanie całości klucza:

procedura: odzyskaj_klucz(j, P)
1 klucz = [];
2 poprzednik = j;
3 **dla:** $i = n_k, n_k - 1, ...1$
4 poprzednik = P[j][poprzednik][i];
5 klucz := [poprzednik]+klucz;
6 **zwróć:** klucz

Dla dowodu poprawności zauważmy tylko, iż niezmiennikiem jest fakt, że po m-tej iteracji (numerowanej od zera) dla $m = 0, 1, 2...n_k - 1$ klucz przechowuje elementy maksymalizującego klucza od $n_k - 1 - m$ do $n_k - 1$, stąd po n_k-tej iteracji będzie przechowywał elementy z pozycji maksymalizującego klucza od 0 do $n_k - 1$, czyli cały klucz.

Ostatecznie możemy więc zapisać cały *algorytm bigramowy.*

Algorytm 4: Algorytm bigramowy

dla: $i, j \in \mathbb{Z}_l$
1 $P_[i][j][1] = i$;
2 $T_1[i][j] = w_2^{(0,1)}(i, j)$;
dla: $d = 2, 3, ...n_k$
3 **dla:** $i, j \in \mathbb{Z}_l$
4 $T_{d+1}[i][j] = \max_{k \in \mathbb{Z}_l} T_d[i, k] + w_2^{(d,d+1)}(k, j)$;
5 $P_[i][j][d + 1] = \arg\max_{k \in \mathbb{Z}_l} T_d[i, k] + w_2^{(d,d+1)}(k, j)$;
zwróć: odzyskaj_klucz($\arg\max_i T_{n_k}[i][i]$, P)

Dla wygody zapisu nie rozdzielaliśmy specjalnie ostatniej iteracji, mimo że wystarczy ją wykonać dla par i, i – wykonanie jej dla wszystkich i, j nie zmienia zresztą istotnie złożoności algorytmu – oczywiście dalej pozostaje on poprawny. Zauważmy teraz, że wystarczy zamienić $\mathbb{Z}_l$ – w miejscach gdzie występuje – na *zbiór wszystkich szyfrów długości jeden* z danej rodziny i dostaniemy poprawne rozwiązanie problemu bigramów dla wszystkich szyfrów (polialfabetycznych podstawieniowych) długości n_k o własności:
wartość zakodowanej litery na pozycji i zależy tylko od samego tekstu oraz szyfru na pozycji i mod n_k

Długość możemy tu rozumieć intuicyjnie jako najmniejszą liczbę elementów S_A, czyli bijekcji nad alfabetem, których znajomość wystarczy do zakodowania/zdekodowania tekstu. Dla 3 rodzajów szyfru o długości n rzeczywiście do zakodowania/zdekodowania była potrzebna i wystarczająca znajomość n bijekcji (w przypadku Vigenère'a i autoklucza były to bijekcje-przesunięcia, w przypadku afinicznego szyfru Vigenère'a były to funkcje postaci $ai + b$). Długość szyfru monoalfabetycznego to 1 itd. Własność powyższa zapewnia nam to, że możemy rozpatrywać wszystkie elementy odpowiadające $i \mod n_k$ jako zaszyfrowane tą samą funkcją ('szyfrem długości 1') i osobno dekodować takie właśnie grupy. Wszystkie rozważane w poprzednim podrozdziale szyfry miały tę własność (dla Vigenère'a i szyfru afinicznego jest to oczywiste, dla szyfru z autokluczem też, gdy przypomnimy sobie postać funkcji szyfrującej i deszyfrującej). Problemem, żeby ten algorytm zadziałał dla każdego szyfru o podanej powyżej własności, jest oczywiście rozmiar podzbioru S_A, po którym iterujemy w celu znalezienia $\arg\max$ w danym kroku – robimy to sprawdzając wszystkie możliwości. W przypadku szyfru z autokluczem i szyfru Vigenère'a jest to l – długość alfabetu. W przypadku szyfru afinicznego $\varphi(l) \cdot l$, gdzie φ jest funkcją Eulera oznaczającą moc $\mathbb{Z}_l^*$ – zbioru liczb względnie pierwszych z l mniejszych od l.
Zanim przejdziemy do dalszej analizy algorytmu, warto przypomnieć o tym, jak kodować i dekodować litery tylko na pozycjach $\equiv i \mod n_k$. Dla szyfrów Vigenère'a i afinicznego jest to proste, dla każdego $j = i, i + n_k, i + 2n_k... < L$, gdzie L jest liczbą liter w tekście:

$$d_j = \sigma_i(e_j),$$

gdzie d_j, e_j są j-tymi literami odkodowania/szyfrogramu. Lub, jeśli klucz jest szyfrujący a nie deszyfrujący, zamieniamy d i e miejscami (σ_i jest oczywiście i-tym wyrazem klucza). Nieco inaczej sprawa ma się z deszyfrowaniem (bo to nas bardziej interesuje) tekstu zakodowanego z autokluczem. W istocie da się dekodować grupy odpowiadające elementom modulo niezależnie. Załóżmy, że

mamy na i-tej pozycji klucza dekodującego $-c$ (czyli innymi słowy odgadliśmy, że klucz kodujący miał na tej pozycji c). Wtedy:

$$\begin{aligned} d_i &= e_i - c \\ d_{i+n_k} &= e_{i+n_k} - d_i \\ &... \\ d_{i+tn_k} &= e_{i+tn_k} - d_{i+(t-1)n_k} \end{aligned}$$

Tak więc rzeczywiście we wszystkich tych szyfrach na pozycje $\equiv i \mod n_k$ wpływa tylko i-ty element szyfru. Log-wagi cząstkowe bigramów liczymy więc w ten sposób:

---

1 $\log w_2^{(i,i+1)}(k_i, k_{i+1}) := 0$;
dla: $j = i, i + n_k, ... < m$
2 `/* możliwe, że używając wcześniej odkodowanych elementów na pozycjach` $\equiv i$, $\equiv i + 1$ `*/`
3 odkoduj(e_j);
4 odkoduj(e_{j+1});
5 b = bigramy_po_odkodowaniu($j, j + 1$);
6 sum_log = suma log-wag(b);
7 $\log w_2^{(i,i+1)}(k_i, k_{i+1})$ += sum_log;
zwróć: $\log w_2^{(i,i+1)}(k_i, k_{i+1})$

---

bigramy powyżej to są po prostu bigramy związane z wagą cząstkową $i, i + 1$ (po odkodowaniu e_i, e_{i+1}). Może to być albo pojedynczy bigram albo para bigramów jeśli litery na pozycjach $i, i+1$ są przedzielone niealfabetycznym ciągiem (P+spacja i spacja+O we wcześniejszym przykładzie). W przypadku Vigenère'a i szyfru afinicznego nie trzeba przechowywać dodatkowych informacji, dla autoklucza za każdym razem trzeba aktualizować klucze które zostaną użyte do dekodowania w następnej iteracji. Ponadto należy dbać, żeby cały czas był dostęp do oryginalnego szyfrogramu, aby następne kroki algorytmu mogły z niego korzystać. Jako że bigramy jesteśmy w stanie pobrać w czasie $O(1)$, podobnie mamy stały dostęp do słownika log-częstości, również odkodowanie zajmuje $O(1)$ (oczywiście przyjmując stały czas pojedynczej operacji mnożenia/dodawania liczb całkowitych i zmiennoprzecinkowych), zatem czas pojedynczej iteracji jest stały. Stąd obliczenie cząstkowej wagi bigramowej zajmuje $O(m/n_k)$ (gdyby n w n-gramie było duże, należałoby jeszcze wziąć je pod uwagę, jednak dla $n = 2$ i $n = 3$ możemy traktować tę wartość jak stałą). Jest to pierwszy element analizy złożoności czasowej. Do kolejnych elementów przejdziemy później, ale najpierw pokażemy jak algorytm zadziałał na naszym szyfrogramie, najpierw dla uprzednio zaszyfrowanego MONTE CARLO MARKOV CHAINZ szyfrem (1,2), czyli szyfrogramu: NQOVF DCSNP NCSMPX EICJPA. W istocie, dostaliśmy poprawne odszyfrowanie: MONTE CARLO MARKOV CHAINZ.

Sprawdźmy jak sobie poradziły obie metody dla szyfru (1,2,3). Szyfrogram to:
AEDIT TOHBD BRFAEK RYOYDO
Metoda monogramowa poprawnie zdekodowała jedną trzecią tekstu, co w tym wypadku nie pomaga zbytnio: RENYU CFHLT CAWAOA SHFYNE Metoda bigramowa natomiast: MONTE CARLO MARKOV CHAINZ. A więc od razu można podejrzewać, że metoda *bigramowa* jest istotnie lepsza od monogramowej.
W tym wypadku nie ma nic za darmo – algorytm bigramowy ma również istotnie większą złożoność czasową i pamięciową, oceńmy więc: w każdej iteracji $d = 1, 2.., n_k$ dla każdej pary i, j (i, j – szyfry długości 1 z danej rodziny) musimy obliczyć max i arg max, a więc i wagę cząstkową s razy, gdzie s jest liczbą szyfrów długości 1 w rodzinie szyfrów, którą rozważamy (w przypadku Vigenère'a jest to po prostu l), trzeba bowiem to zrobić dla wszystkich k, po których przebiegają max, arg max (zwróćmy uwagę, że możemy i powinniśmy obliczyć oba wewnątrz tej samej procedury). Każda taka iteracja zajmuje więc s^2 – liczba par i, j, razy s – liczba argumentów, po których przebiega krok razy $O(m/n_k)$ – koszt obliczenia wagi. Łącznie dostajemy więc złożoność czasową $O(s^3 \cdot m)$, natomiast złożoność pamięciowa zdominowana jest przez tablicę *poprzedników* (zauważmy, że do przechowywania wartości log-wag wystarczą nam tak naprawdę dwie tablice *aktualna* i *przyszła*, z tablicą poprzedników tak nie jest, bo potrzebujemy całej 'ścieżki' klucza). Tak więc używamy dodatkowej pamięci $O(s^2 \cdot n_k)$. W zależności od s i m $O(s^3 \cdot m)$ to może być dużo lub mało, ale

często może się zdarzyć, że jest to naprawdę dużo, przez co w pewnych wypadkach nie jesteśmy w stanie w rozsądnym czasie przeprowadzić skutecznego ataku kryptograficznego. Wtedy z pomocą przychodzą nam metody MCMC, ale o tym opowiemy później.
Teraz możemy się jeszcze zastanowić, jak uogólnić ideę algorytmu bigramowego na dowolne n-gramy. Nie jest to trudne – wystarczy zastosować ten sam argmax-owy mechanizm, tyle tylko, że dla spójności musimy brać pod uwagę klucze długości $n-1$ – po takich będzie iterowało k dla ustalenia max i arg max (w przypadku $n = 2$, było to – jak widzieliśmy – 1, w przypadku $n = 1$ było to 0, kolejne 'cząstki' w ogóle od siebie nie zależały), bo taki rozmiar jest wymagany dla utrzymania spójności (np. dla trigramów by obliczyć maksymalne wagi dla pozycji (0,1,2,3,4), mając je już dla (0,1,2,3) musimy pamiętać o spójności na pozycjach (2,3)). Tak więc złożoność czasowa podnosi się do $O(s^n \cdot s^{n-1} \cdot m) = O(s^{2n-1}m)$, a pamięciowa – $O(s^{2(n-1)} \cdot n_k)$. Przy już dość małym n więc (już dla szyfru Vigenère'a, alfabetu o długości $l = 26$ i $n = 3$ oraz tekstu długości, powiedzmy 1000), stosowanie n-gramowej taktyki staje się zupełnie nieefektywne czasowo i pamięciowo – w deterministycznej wersji. Już dla $n = 2$ jest natomiast *wolne*.
Algorytm również dalej naturalnie rozszerza się na *wagi mieszane* – rozpatrujemy wtedy po prostu N-gramy będące NWW wszystkich n wziętych z niezerowymi współczynnikami (tak że NWW wchodzi wtedy do złożoności zamiast n). Zwróćmy jeszcze uwagę, że dla dużych n liczby n-gramów związanych z daną pozycją nie powinno się już traktować jak stałej – może stać się ona istotną częścią złożoności. Liczbą n-gramów związanych z daną pozycją jest $\sim n$. Tak więc złożoność czasową można zapisać jako $O(n \cdot s^{2n-1} \cdot m)$.

W zakończonym podrozdziale zaproponowałem algorytm deterministyczny rozwiązujący postawiony problem i zaprezentowałem działanie tego podejścia na krótkim napisie. Głównym wkładem tych rozważań jest jednak to, że możemy dzięki deterministycznemu algorytmowi przeprowadzić testy sprawdzające skuteczność arg max-owej metody (kiedy w ogóle może dać nam satysfakcjonujący wynik), czym zajmę się w kolejnym podrozdziale (w zasadzie *podpodrozdziale*).

4.4.2 Ocena skuteczności

Chcemy sprawdzić, kiedy rozważane uprzednio podejście ma w ogóle sens. Można intuicyjnie wyczuć, że im większa długość szyfru w stosunku do długości szyfrowanego tekstu, tym szyfrowanie jest bezpieczniejsze. Rzeczywiście, wtedy szyfr coraz bardziej zbliża się do szyfru Vernama – który jest *szyfrem nie do złamania* – jeśli tylko szyfrowane są wszystkie znaki i klucz jest generowany losowo jednostajnie [32]. Szyfr Vernama to odmiana szyfru Vigenère'a – klucz jednak ma długość taką jak oryginalna wiadomość, przez co, jeśli klucz jest wygenerowany zgodnie z rozkładem jednostajnym, nie jesteśmy w stanie odczytać poprawnie żadnych zależności statystycznych z szyfrogramu – każda litera wiadomości jest przesunięta o losową wylosowaną jednostajnie wartość – więc samą literę szyfrogramu można traktować jako wylosowaną z alfabetu z rozkładem jednostajnym! Każda sekwencja liter jest więc tak samo prawdopodobna niezależnie od rzeczywistego tekstu. W bardzo małym stopniu jest możliwe zaatakować ten szyfr, jeśli występują znaki niekodowane, np. załóżmy, że mamy niekodowaną spację i zakodowany szyfrem Vernama tekst ABC DEF. Możemy wtedy badać najbardziej prawdopodobne sekwencje dwóch słów w angielskim – jednak bez dodatkowej znajomości kontekstu jest to w zasadzie dalej niemożliwe, tzn. HEY BOY jest praktycznie tak samo prawdopodobne jak YES SIR, jeśli nie wiemy np., że to list dziewczyny do chłopaka albo odwrotnie – fragment scenariusza do filmu wojennego. Bez znajomości kontekstu natomiast, przy użyciu zautomatyzowanych metod, szyfr nadal pozostaje *praktycznie* nie do złamania.
Skupiamy się nadal na metodach n-gramowych. Dzięki wyprowadzeniu algorytmów deterministycznych możemy zaimplementować je i sprawdzić, kiedy poziom odszyfrowania otrzymany w ten sposób jest zadowalający, innymi słowy, sprawdzić górne ograniczenia tej metody i wszystkich opartych na tym schemacie. Zwróćmy uwagę, że dla każdego 1000-elementowego zaszyfrowanego tekstu przy kluczu długości 1000 (szyfr Vernama) i niekodowanych (bądź ich braku) znakach w tych samych miejscach algorytmy deterministyczne dadzą (muszą dać!) ten sam wynik, po prostu najbardziej prawdopodobną (względem przyjętego rozkładu znormalizowanych w) sekwencję znaków w języku angielskim – skoro szyfr jest nie do złamania, to jak mielibyśmy go złamać? Do pewnego momentu jednak jesteśmy w stanie taki szyfr całkiem skutecznie zaatakować. Jednak klasycznie opisywana metoda analizy części zaszyfrowanych monoalfabetycznie wg częstości pojedynczych liter dla długiego klucza zawodzi. Już we wcześniejszym rozdziale na podstawie krótkiego

napisu pokazaliśmy, że jeśli ciągi zaszyfrowane monoalfabetycznie są krótkie, to w bardzo nikłym stopniu wyrażają się tam statystyczne zależności. Nawet jeśli uda się poprawnie zdekodować któryś z monoalfabetycznych ciągów, to ponieważ liczba pól do zdekodowania jest duża, niewiele nas to porusza do przodu. Powiedzmy, że klucz ma 1/4 długości tekstu – monoalfabetyczne ciągi mają długość 4 – to bardzo mało (daje to niski *indeks koincydencji* [34]). Jeśliby nawet podobnie jak dla szyfru Cezara założyć, że na każdym z tych ciągów najczęściej występowała jedna z czterech najpopularniejszych liter alfabetu, nadal jest do sprawdzenia wykładniczo wiele wartości ($4^{m/4}$, m – długość tekstu). Pomocą są tu wyprowadzone metody, które poza samymi monoalfabetycznie zaszyfrowanymi ciągami biorą też pod uwagę zależności między nimi.

Przeprowadziłem testy dla szyfru Vigenère'a z klasycznym 26-znakowym alfabetem, żeby to zobrazować, a także, żeby dać pewne wyobrażenie o granicach skuteczności metod bigramowych i monogramowych. Testy będą przeprowadzane przy użyciu alfabetu Vigenère'a (duże angielskie litery), szyfrując i deszyfrując, zachowujemy niekodowane znaki – jak w klasycznym opisie szyfru Vigenère'a, który znajdziemy na polskiej Wikipedii [11]. Jak już wcześniej wspomniałem, słownik log-częstości uni- (mono-) bi- (di-) i trigramów wygenerowałem na podstawie trzech książek (w języku angielskim): *Duma i uprzedzenie* Jane Austen, *Oliver Twist* Karola Dickensa oraz *Wojna i pokój* Lwa Tołstoja pobranych z [31]. Klucze szyfrujące generujemy losowo (przy użyciu metod z biblioteki *random* w Pythonie – po prostu losowo generujemy każdą kolejną współrzędną). Test przeprowadzamy na losowo wybranych fragmentach długości m książki *Rok 1984* George'a Orwella (gdzie oczywiście małe litery są wcześniej skonwertowane na wielkie, podobnie tekst jest wcześniej oczyszczony z podwójnych spacji itd. – mogłaby to być równie dobrze część działania algorytmu dekodującego, nie robi to różnicy). Skuteczność dekodowania jest dana stosunkiem zaszyfrowanych pól, które udało się odszyfrować poprawnie do wszystkich zaszyfrowanych pól. I tak dostajemy skuteczności:

	SKUTECZNOŚĆ	
DŁUGOŚĆ KLUCZA	METODA MONOGRAMOWA	METODA BIGRAMOWA
1	1.0	1.0
2	1.0	1.0
3	0.66	1.0
4	0.5	1.0
5	0.2	0.4
6	0.33	0.83
7	0.28	0.14
8	0.25	0.5
9	0.11	0.33
10	0.3	0.4
11	0.09	0.27
12	0.25	0.41
13	0.23	0.0
14	0.14	0.28
15	0.2	0.26
16	0.12	0.18
17	0.17	0.23
18	0.22	0.27
19	0.15	0.21
20	0.15	0.2

Tabela 2: Skuteczność metod n-gramowych dla fragmentu *Roku 1984* o dł. 20

	SKUTECZNOŚĆ	
DŁUGOŚĆ KLUCZA	METODA MONOGRAMOWA	METODA BIGRAMOWA

1	1.0	1.0
2	1.0	1.0
3	0.66	1.0
4	0.75	1.0
5	0.6	1.0
6	0.5	1.0
7	0.85	1.0
8	0.5	0.87
9	0.55	0.88
10	0.5	1.0
11	0.45	0.81
12	0.33	0.66
13	0.30	0.46
14	0.42	0.64
15	0.26	0.66
16	0.37	0.56
17	0.29	0.41
18	0.27	0.55
19	0.31	0.52
20	0.15	0.4
21	0.23	0.28
22	0.18	0.13
23	0.13	0.30
24	0.20	0.37
25	0.24	0.32

Tabela 3: Skuteczność metod n-gramowych dla fragmentu *Roku 1984* o dł. 50

	SKUTECZNOŚĆ	
DŁUGOŚĆ KLUCZA	METODA MONOGRAMOWA	METODA BIGRAMOWA
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	1.0	1.0
5	0.8	1.0
6	0.83	1.0
7	0.85	1.0
8	0.75	1.0
9	0.88	1.0
10	0.6	1.0
11	0.63	0.90
12	0.5	1.0
13	0.53	1.0
14	0.71	0.92
15	0.53	0.86
16	0.56	0.93
17	0.41	1.0
18	0.38	0.77
19	0.36	0.89
20	0.6	0.9
21	0.47	0.80
22	0.27	0.54
23	0.39	0.78

24	0.25	0.62
25	0.36	0.72
26	0.15	0.73
27	0.33	0.59
28	0.39	0.57
29	0.27	0.55
30	0.33	0.5
31	0.25	0.51
32	0.37	0.59
33	0.21	0.36
34	0.23	0.44
35	0.37	0.68
36	0.27	0.33
37	0.24	0.35
38	0.28	0.5
39	0.28	0.38
40	0.22	0.27
41	0.19	0.41
42	0.23	0.45
43	0.23	0.18
44	0.22	0.20
45	0.15	0.26
46	0.17	0.28
47	0.17	0.21
48	0.14	0.27
49	0.14	0.20

Tabela 4: Skuteczność metod n-gramowych dla fragmentu *Roku 1984* o dł. 100

	SKUTECZNOŚĆ	
DŁUGOŚĆ KLUCZA	METODA MONOGRAMOWA	METODA BIGRAMOWA
20	0.95	1.0
40	0.82	1.0
60	0.58	1.0
80	0.6	0.93
100	0.37	0.8
120	0.35	0.72
140	0.29	0.55
160	0.26	0.46
180	0.21	0.36
200	0.23	0.30
220	0.20	0.28
240	0.2	0.26
260	0.17	0.20
280	0.13	0.19
300	0.16	0.17

Tabela 5: Skuteczność metod n-gramowych dla fragmentu *Roku 1984* o dł. 500

	SKUTECZNOŚĆ	
DŁUGOŚĆ KLUCZA	METODA MONOGRAMOWA	METODA BIGRAMOWA

20	1.0	1.0
40	1.0	1.0
60	0.83	1.0
80	0.76	1.0
100	0.64	1.0
120	0.58	1.0
140	0.52	0.92
160	0.47	0.89
180	0.33	0.87
200	0.36	0.85
220	0.35	0.73
240	0.3	0.67
260	0.28	0.54
280	0.30	0.53
300	0.23	0.43
320	0.29	0.46
340	0.25	0.39
360	0.20	0.31
380	0.21	0.30
400	0.24	0.29
420	0.2	0.27
440	0.20	0.30
460	0.18	0.24
480	0.18	0.23
500	0.18	0.24
520	0.17	0.23
540	0.16	0.20
560	0.16	0.18
580	0.17	0.18

Tabela 6: Skuteczność metod n-gramowych dla fragmentu *Roku 1984* o dł. 1000

	SKUTECZNOŚĆ	
DŁUGOŚĆ KLUCZA	METODA MONOGRAMOWA	METODA BIGRAMOWA
50	1.0	1.0
100	0.95	1.0
150	0.77	1.0
200	0.64	0.99
250	0.54	0.98
300	0.46	0.95
350	0.41	0.87
400	0.36	0.81
450	0.34	0.67
500	0.31	0.64
550	0.28	0.58
600	0.25	0.51
650	0.25	0.40
700	0.22	0.38
750	0.21	0.31
800	0.19	0.31
850	0.19	0.29
900	0.16	0.27
950	0.14	0.26

1000	0.13	0.22

Tabela 7: Skuteczność metod n-gramowych dla fragmentu *Roku 1984* o dł. 2000

	SKUTECZNOŚĆ	
DŁUGOŚĆ KLUCZA	METODA MONOGRAMOWA	METODA BIGRAMOWA
500	0.92	1.0
1000	0.62	0.99
1500	0.44	0.93
2000	0.36	0.77
2500	0.28	0.61
3000	0.25	0.46
3500	0.21	0.39
4000	0.19	0.31
4500	0.17	0.27

Tabela 8: Skuteczność metod n-gramowych dla fragmentu *Roku 1984* o dł. 10000

Wyniki są powtarzalne – metoda bigramowa daje znacznie lepsze rezultaty niż monogramowa. Począwszy od kluczy długości ok. 1/10 - 1/8 długości tekstu są to już wyniki ok. 2 razy lepsze. Bigramowa metoda zachowuje skuteczność na poziomie średnio >50% do kluczy o długości ok. 1/4 - 1/3 długości tekstu. W ewentualnej kontynuacji pracy można by się pokusić o bardziej systematyczne testy diagnostyczne, tutaj przede wszystkim prezentuję ideę, więc zostajemy na razie przy tych zgrubnych oszacowaniach, które dają nam jednak wystarczającą wiedzę do przyjętych celów. Jedno jest pewne, metoda bigramowa jest skuteczna dla znacznie większego zbioru długości kluczy, więc to ona (albo 'większa') powinna być bazą naszego ataku.
Niestety ze względu na dużą złożoność czasową, jak opisałem w poprzednim podrozdziale, nie było możliwości przetestowania metody trigramowej (ani dla większych n). Również testy metody bigramowej zajmowały – zwłaszcza dla tekstów o długości większej niż 1000 – bardzo dużo czasu. Załóżmy teraz, że tego czasu nie mamy, potrzebujemy zdekodować wiadomość *szybko* i *dostatecznie* dobrze. Jest parę powodów, dla których możemy tego potrzebować, czasem potencjalna złożoność czasowa problemu może być tak duża, że realizacja tego algorytmu nie jest realna na komputerze, który mamy do dyspozycji: rozszerzony szyfr Vigenère'a dla alfabetu 93-symbolowego, na którym później również (poza klasycznym o dł. 26) będziemy pracować, zadaje na pojedynczej współrzędnej 5580 możliwości! Złożoność czasowa algorytmu bigramowego dla tekstu długości 1000 jest więc rzędu: 1.7374111×10^{13}, ponadto dodatkową potrzebną pamięć dla szyfru długości 200 można oszacować na 6.22728 GB. Tak naprawdę na komputerze, na którym pracowałem (HP ProBook), już obliczenia dla 26-elementowego alfabetu zajmowały bardzo długo (testy dla klucza o długości 2000 zajęły kilka godzin – zauważmy, że stała oznaczająca czas wewnątrz iteracji, również nie jest aż tak mała). Nie jest to dobre również wtedy, kiedy szybko chcemy sprawdzić, czy próba takiego ataku ma w ogóle szansę – gdy nie wiemy, jakim szyfrem tekst został zakodowany i chcemy wypróbować tę możliwość, ale nie tracić na nią niepotrzebnie zbyt dużo czasu i zasobów albo gdy wynik dekodowania jest nam potrzebny na zaraz. Może być też po prostu, że szybko tracimy cierpliwość – powodów jest wiele.

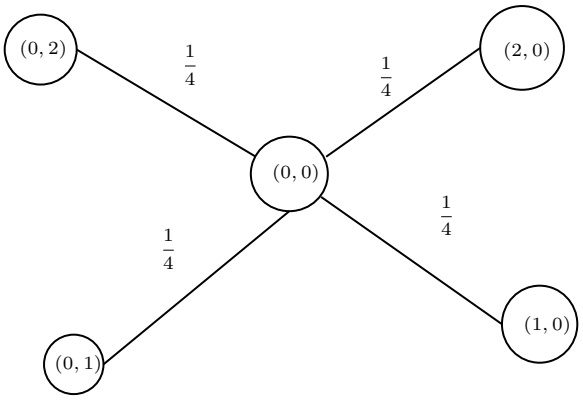
Tak więc podsumowując, metoda monogramowa szybko przestaje działać, metoda bigramowa i dalsze (intuicyjnie) dłużej zachowują skuteczność, ale mogą okazać się za wolne. Jak zachować dobre wyniki, jednocześnie przyspieszając algorytm? Odpowiedź dają metody Markov Chain Monte Carlo.

4.5 Algorytmy MCMC

Zanim zaczniemy rozwijać opis algorytmu, przydatne będzie przedstawienie idei MCMC w postaci grafu (sam graf pojawi się jeszcze wprost jako centrum problemu komiwojażera) odpowiadającego macierzy generującej kandydatów (czy też krawędziom między wierzchołkami, między którymi jest niezerowe prawdopodobieństwo przejścia). Zakładam, że to pojęcie jest znane, przypomnę tylko parę koncepcji, które się później jeszcze przydadzą przy okazji analizy *mixing time*. Odległością między wierzchołkami nazwiemy długość (liczbę krawędzi) najkrótszej drogi (przechodzącej przez jak najmniej krawędzi) między tymi wierzchołkami. Odległość wierzchołka do samego siebie to zero, do wierzchołków połączonych z nim krawędzią to jeden itd. Średnicą w grafie nazwiemy maksymalną odległość w grafie. Gdy mamy funkcję ze zbioru wierzchołków grafu w $\mathbb{R}$, można też mówić o maksimach i minimach lokalnych względem tej funkcji – gdy wartość funkcji dla wszystkich sąsiadów jest odpowiednio nie większa bądź nie mniejsza od wartości w danym wierzchołku. Tak więc analogicznie maksimum lokalnym funkcji $S \to \mathbb{R}$ dla łańcucha Markowa o przestrzeni stanów S nazywamy stan, taki że na wszystkich stanach sąsiednich (czyli takich, do których rozważany stan ma niezerowe prawdopodobieństwo przejścia) funkcja przyjmuje nie większą wartość (analogicznie definiujemy minimum).
Wracamy więc do problemu dekodowania. Jak poprzednio, zaprezentuję podejście dla algorytmu Vigenère'a w wersji standardowej, koncept naturalnie rozszerza się na pozostałe warianty.

Przypomnijmy sobie, że szyfr (kodujący czy dekodujący) Vigenère'a i opisane szyfry pochodne długości n_k przedstawiają się za pomocą n_k szyfrów pojedynczych (konkretnie n_k elementów $\mathbb{Z}_l$). Chcemy poszukać $\arg\max$ dla n-gramowej wagi, ale metoda deterministyczna – przynajmniej ta, którą znamy – jest dla nas za wolna. Metody gradientowe czy numeryczne poszukiwania maksimów nie posłużą nam tu, bo nie za bardzo da się oczekiwać, żeby nasza *waga* zachowywała się jakkolwiek regularnie. Przechodzimy więc do metod probabilistycznych. Będziemy chcieli więc symulować jakoś rozkład *a posteriori* (znormalizowane w) na zbiorze kluczy, tak że po pewnym czasie powinniśmy trafić w nasze poszukiwane maksimum. Oczywiście mamy do czynienia z sytuacją taką jak opisaną we wcześniejszym rozdziale. Rozkład jest skomplikowany, nieznany *wprost* – mamy tylko informację jak go obliczać i że ma być proporcjonalny do *wagi* n-gramowej (gra słowna waga-gram nie była moim zamierzeniem – powstała przez przypadek). Nie jest to dla nas problemem dzięki temu, że dysponujemy pięknym algorytmem Metropolisa-Hastingsa.

Żeby być w stanie go zastosować musimy najpierw opisać jak będziemy próbowali wędrować po zbiorze kluczy dekodujących, czyli opisać macierz generującą kandydatów $\mathbb{Q}$. Macierz powinna mieć możliwie prostą postać – kolejne przejścia powinny być szybkie! Pomysł jest prosty i przypomina błądzenie po n_k-wymiarowej kostce. Mianowicie, przypomnijmy sobie, ze nasze klucze przedstawia się po prostu jako n_k 'kluczowych' współrzędnych. Możemy więc po prostu próbować zmieniać jedną współrzędną w jednym kroku – wybór współrzędnej i zmiany zgodnie z rozkładem jednostajnym – i akceptować bądź odrzucać zmianę zgodnie z funkcją *wagi*. Przykładowo, poniżej można zobaczyć wybór kandydatów dla stanu $(0, 0)$ przy kluczu Vigenère'a o długości 2 i alfabecie długości 3.



Rysunek 1: Fragment macierzy generującej kandydatów przedstawiony przy pomocy grafu

Ogólnie więc przyjmujemy (niech s będzie liczbą kluczy długości 1 dla rozważanej rodziny):

$$q_{\boldsymbol{ij}} = \begin{cases} \frac{1}{(s-1)n_k} \text{ jeśli } \boldsymbol{i} \text{ różni się od } \boldsymbol{j} \text{ na dokładnie jednej współrzędnej} \\ 0 \text{ w przeciwnym razie.} \end{cases}$$

Taki wybór kandydatów ma pewną cechę, której zazwyczaj pożądamy: zachodzi jakaś wersja *ciągłości*. To znaczy: łańcuch zmienia się nie skokowo, ale raczej drobnymi krokami, co porządkuje wędrówkę. Ponadto ułatwia obliczanie update'ów wagi (funkcji stanu). Zanim przejdziemy dalej wypada (1) zauważyć, że macierz jest nieredukowalna – rzeczywiście aby dojść z jednego stanu (klucza) do drugiego po prostu kolejno zmieniamy różniące się między stanami współrzędne – zapewnia nam to nieredukowalność wynikowego łańcucha; (2) łańcuch wynikowy będzie nieokresowy – klucz odpowiadający maksymalnej wadze będzie z niezerowym prawdopodobieństwem odrzucał kandydata (jeśli tylko rozkład stacjonarny nie jest jednostajny, zakładamy, że nie jest, język angielski ma bowiem pewną uporządkowaną strukturę). Następnie postępujemy zgodnie z algorytmem Metropolisa-Hastingsa – a w zasadzie Metropolisa, zwróćmy bowiem uwagę, że macierz generująca jest symetryczna. Ponadto przypomnijmy sobie, że obliczanie maksymalnego klucza ze względu na monogramy jest szybkie, a może on być dobrym punktem startowym, bowiem prawdopodobnie część pól klucza (nawet jeśli niewielka) jest wtedy odgadnięta poprawnie. Ogólnie uznajemy, że stan startowy podlega wyborowi według ustalonego przez nas kryterium – dla algorytmu bigramowego będziemy przyjmować kryterium maksymalnego klucza ze względu na monogramy, dla trigramowego – opiszę to przy okazji rozdziału z wynikami. Wtedy algorytm ma postać:

Algorytm 5: Algorytm n-gramowy MCMC

```
1  maks_stan:=stan_aktualny:=ustalony_startowy;
2  odszyfrowany_tekst := odszyfruj(szyfrogram, stan_aktualny);
3  maks_log_waga := aktualna_log_waga := log_waga(stan_aktualny);
   dla: i = 1, 2, ...,liczba_kroków
4    kandydat := generuj_kandydata(stan_aktualny);
5    r := różnica_log_wag(kandydat, stan_aktualny);
6    U ~ U(0,1), log_u = log(U)
7    jeśli: r > log_u
8      odszyfrowany_tekst := update(odszyfrowany_tekst, stan_aktualny, kandydat);
9      stan_aktualny := kandydat;
10     aktualna_log_waga := aktualna_log_waga + r;
11     jeśli: aktualna_log_waga > maks_log_waga
12       maks_stan:=stan_aktualny;
13       maks_log_waga := aktualna_log_waga;
   zwróć: maks_stan
```

Omówmy teraz nieco dokładniej implementację poszczególnych kroków. By móc skutecznie i szybko obliczać *log-wagi* (n-gramowe), trzymamy w słowniku n-gramów aktualne informacje: ile razy n-gram wystąpił w danym odszyfrowaniu (za pomocą aktualnego stanu/klucza). Korzystając z tego i drugiego wzoru na *wagę*

$$\log w_n(\boldsymbol{k}) = \sum_{g \in G} r_D(g) f_n(g),$$

jesteśmy w stanie najpierw obliczyć log-wagę stanu początkowego (r bierzemy z właśnie wspomnianego słownika). Następnie, gdy dostajemy kandydata, generujemy różnicę log-wag. Korzystamy tu z faktu, że zmieniamy klucz tylko na jednej współrzędnej, więc zgodnie z rozważaniami z poprzedniego rozdziału możemy brać pod uwagę tylko te n-gramy, które mają w sobie pozycję równą *modulo* zmienianej pozycji klucza (w każdej grupie kolejnych n_k znaków będzie tylko 1 taki monogram, 2 takie bigramy itd.). Na ich podstawie wyliczamy słownik różnic wystąpień n-gramów między aktualnym odszyfrowaniem, a potencjalnym odszyfrowaniem przez kandydata. Na tej podstawie możemy wyliczyć też różnicę wag, jeśli liczba wystąpień g zmieniła się o $c(g)$, różnica log-wag będzie wynosić :

$$\Delta_{w_n}(\boldsymbol{i}, \boldsymbol{j}) = \sum_{\text{zmienione } g} c(g) f_n(g).$$

Iterujemy oczywiście tylko po *gramach* powiązanych ze zmienioną współrzędną kandydata (tylko te należą do słownika różnic wystąpień). Następnie, zgodnie z algorytmem Metropolisa (po na-

łożeniu na obie strony nierówności logarytmu) przechodzimy z aktualnego stanu (klucza) i do stanu-kandydata j, jeśli:

$$\Delta_{w_n}(\boldsymbol{i}, \boldsymbol{j}) = \log(w_n(\boldsymbol{j})) - \log(w_n(\boldsymbol{i})) > log(U),$$

gdzie U jest oczywiście z rozkładu jednostajnego na $[0, 1]$. Update'ujemy aktualną wagę i aktualny słownik (dodajemy do wpisów słownika wystąpień wpisy ze słownika różnic). Jeśli nowa waga jest większa niż największa dotychczas, aktualizujemy maksymalną wagę i maksymalizujący klucz. Na koniec zwracamy klucz dekodujący, który zmaksymalizował wartość wagi w trakcie wędrówki łańcucha – widać, że nie bierzemy tu ostatniego odwiedzonego *stanu*, tylko najlepszy – pomysł przejęty od [3]. W ten sposób nie musimy się martwić o kryteria zbieżności – arbitralnie po prostu ustalamy liczbę kroków i mamy nadzieję, że w tym czasie łańcuch odwiedzi maksymalizujący klucz. Chcę ocenić liczbę kroków, która jest niezbędna do tych odwiedzin (mając pewne heurystyczne podejrzenia). Dalej w praktyce można wprowadzać 'zbieżnościowe' rozwiązania, aby łańcuch nie kontynuował niepotrzebnie swojego przebiegu, takie jak procent odrzucanych kandydatów (gdy kandydaci są odrzucani bardzo często albo wręcz cały czas, to znak, że łańcuch osiągnął stabilność – trafił na maksimum lokalne (możemy mieć nadzieję, że również globalne)), można też oczekiwać pewnej wartości funkcji stanu (log-wagi), która nas satysfakcjonuje itd. W wersji zbieżnościowej po spełnieniu pewnego kryterium 'zatrzymujemy łańcuch'. W części pracy dotyczącej dekodowania jednak nie będę się na tym skupiał. Nieco więcej będzie wspomniane na ten temat przy okazji problemu komiwojażera – stosowane tam symulowane wyżarzanie jest metodą, która bardziej skupia się na zbieżności (sama z siebie ogranicza też 'ruchliwość' łańcucha wraz z upływem czasu).

Możemy więc oszacować złożoność czasową – najpierw kroku początkowego: tu musimy odkodować tekst ($O(m)$ – m długość tekstu), wyliczyć słownik i wagę (również $O(m)$, bo tyle jest n-gramów). Następnie krok: wyliczenie różnicy to $O(m/n_k)$ – zakładamy, że n (od n-gramów) jest małe, co najwyżej 3, więc może być potraktowane jako stała. Podobnie wyliczenie *update'u* wagi i potencjalny *update* odszyfrowania potrwają $O(m/n_k)$, pozostałe części kroku mają stałą złożoność. Ogólnie więc mamy złożoność $O(m)$ dodać $O(m/n_k \cdot steps)$, przy czym liczba kroków jest raczej większa niż n_k, więc drugi składnik dominuje.
Odnośnie złożoności pamięciowej, wygląda to z grubsza tak, że potrzebujemy kopii szyfrogramu (którą próbujemy dekodować) i trzech kluczy: najlepszego, aktualnego i kandydata (w praktyce wystarczy nam informacja o zmianie, którą wnosi kandydat). Dodatkowa pamięć jest więc zdominowana przez kopię szyfrogramu, czyli $O(m)$.

Można zauważyć, że jeśli tylko nie będzie potrzebna zbyt duża liczba kroków, jest szansa na osiągnięcie dużej poprawy w złożoności czasowej i pamięciowej. Pozostaje odpowiedzieć sobie na pytanie: ile kroków nam wystarczy, by osiągnąć oczekiwane wyniki? Tym zajmuje się następny podrozdział. Zagadnienie to jest zagadnieniem trudnym, nie da się go rozwiązać w ogólnym przypadku, bo w pewnym sensie ogólny przypadek nie istnieje – każdy szyfrogram, a więc i rozkład a posteriori jest inny, więc analiza będzie raczej heurystyczna, natomiast będziemy mogli zbadać stopień jej poprawności doświadczalnie.

4.6 Zbieżność, mixing time, analiza heurystyczna

Chcielibyśmy wiedzieć, jak szybko zostanie osiągnięty stan (klucz) maksymalizujący po starcie ze stanu (klucza) początkowego. Przy oczekiwanej własności, że wszystkie stany o dużych wagach skupiają się w sąsiedztwie (według sąsiedztwa definiowanego przy pomocy *macierzy generującej kandydatów*) $\arg\max$ jest to niemal równoważne z badaniem zbieżności łańcucha do rozkładu stacjonarnego. Przypomnijmy sobie, że nasza macierz generująca kandydatów jest w zasadzie spacerem po hiperkostce – tylko że jest to hiperkostka z s (liczba pojedynczych kluczy) możliwościami na każdej współrzędnej (zamiast $(0, 1)$ czy $(-1, 1)$). Oszacujemy najpierw tzw. *mixing time* na przestrzeni szyfrów s^{n_k}, ale z jednostajnym prawdopodobieństwem dla każdego przejścia – tj. krok polega na tym, że jednostajnie losujemy współrzędną i tam losujemy nową wartość współrzędnej zgodnie z rozkładem jednostajnym na $1, 2...s$ (numerujemy klucze długości 1). Zacznijmy od tego, czym jest *mixing time*. Wracamy do klasycznych oznaczeń S na przestrzeń stanów i $\mathbb{P}$ na macierz przejścia. Wcześniej przypomnijmy sobie, że dla dowolnego stanu i z przestrzeni stanów mamy, że:

$$d(t) = \max_{\boldsymbol{i} \in S} \|\boldsymbol{\delta_i}\mathbb{P}^t - \boldsymbol{\pi}\mathbb{P}^t\|_{TV}$$

jest ograniczone z góry przez ciąg dążący monotonicznie do zera. Tak więc można zdefiniować:

$$t_{mix}(\varepsilon) = \inf_t\{t:\ d(t) < \varepsilon\},$$

co nazwiemy *mixing time* na poziomie epsilon. Mówi nam on, że po tym właśnie czasie niezależnie od rozkładu startowego JŁM o macierzy przejścia $\mathbb{P}$ jest od rozkładu stacjonarnego bliżej niż ϵ. Dla zobrazowania pewnej idei, a także z tego powodu, że dowód jest po prostu ładny (bardziej zaawansowane rozważania na temat spacerów po hiperkostce można znaleźć np. w [20]), pokażemy górne ograniczenie na $d(t)$ na takiej samej s-hiperkostce wymiaru n_k, tyle tylko że ze spacerem losowym wg rozkładu jednostajnego (zapominamy na chwilę o naszych wagach). Najpierw zauważmy, że:

$$d(t) \leq \max_{\boldsymbol{i},\boldsymbol{j}\in S} \|\boldsymbol{\delta_i}\mathbb{P}^t - \boldsymbol{\delta_j}\mathbb{P}^t\|_{TV}.$$

Nierówność łatwo dowodzimy podobnie jak w rozdziale z dowodem twierdzenia ergodycznego $\boldsymbol{\pi}\mathbb{P}^t$ – rozkład stacjonarny możemy zapisać jako ważoną sumę $\boldsymbol{\delta_k}\mathbb{P}^t$. Dalej korzystamy z nierówności trójkąta, zakładając, że k był elementem, który zadawał max w równaniu na $d(t)$, dostając:

$$d(t) = \|\boldsymbol{\delta_k}\mathbb{P}^t - \boldsymbol{\pi}\mathbb{P}^t\|_{TV} = \|\boldsymbol{\delta_k}\mathbb{P}^t - \left(\sum_{\boldsymbol{i}\in S}\alpha_i\boldsymbol{\delta_i}\right)\mathbb{P}^t\|_{TV} \leq \sum_{\boldsymbol{i}\in S}\alpha_i\|\boldsymbol{\delta_k}\mathbb{P}^t - \boldsymbol{\delta_i}\mathbb{P}^t\|_{TV} \leq \max_{\boldsymbol{i},\boldsymbol{j}\in S}\|\boldsymbol{\delta_i}\mathbb{P}^t - \boldsymbol{\delta_j}\mathbb{P}^t\|_{TV}.$$

Ostatnia równość wynika oczywiście z tego, że α_i sumują się do jeden. Możemy więc skonstruować coupling, który da nam górne ograniczenie na $d(t)$ dla naszego przypadku. Mianowicie, niech X_n i Y_n startują z dwóch różnych stanów $\boldsymbol{i}$ i $\boldsymbol{j}$. *Coupling* konstruujemy w ten sposób, że dla obu tych kopii łańcucha następny stan losujemy w następujący sposób: najpierw jednostajnie losujemy współrzędną, potem jednostajnie jej nową wartość – tę samą dla obu łańcuchów. Widać więc, że od tego momentu łańcuchy biegną razem na danej współrzędnej. Kiedy trafimy we wszystkie różniące się współrzędne, łańcuchy są więc już sparowane. Oznaczmy pierwszy moment sparowania przez $\tau_{\boldsymbol{i},\boldsymbol{j}}$. Pamiętamy o nierównościach z podrozdziału (2.3), korzystając z tych nierówności a dodatkowo z nierówności Czebyszewa, mamy więc:

$$\begin{aligned} d(t) &= \max_{\boldsymbol{i}\in S}\|\boldsymbol{\delta_i}\mathbb{P}^t - \boldsymbol{\pi}\mathbb{P}^t\|_{TV} \\ &\leq \max_{\boldsymbol{i},\boldsymbol{j}\in S}\|\boldsymbol{\delta_i}\mathbb{P}^t - \boldsymbol{\delta_j}\mathbb{P}^t\|_{TV} \\ &\leq \max_{\boldsymbol{i},\boldsymbol{j}\in S} P(X_t \neq Y_t) = \max_{\boldsymbol{i},\boldsymbol{j}\in S} P(\tau_{\boldsymbol{i},\boldsymbol{j}} > t) \\ &\leq \max_{\boldsymbol{i},\boldsymbol{j}\in S}\frac{\mathbb{E}(\tau_{\boldsymbol{i},\boldsymbol{j}})}{t} \end{aligned}$$

Ponieważ τ odpowiada czasowi trafienia we wszystkie różniące się współrzędne, więc największą wartość przyjmie wtedy, gdy wszystkie n_k współrzędnych będzie się różnić (elementy realizują wtedy średnicę grafu zadanego przez macierz generującą). By obliczyć wartość oczekiwaną dla τ, przeprowadzamy rozumowanie jak przy problemie kolekcjonera kuponów. Ustalmy $\boldsymbol{i},\boldsymbol{j}$, takie co różnią się na wszystkich n_k współrzędnych. przez τ_m oznaczmy moment, gdy trafiliśmy już w m współrzędnych spośród wszystkich n_k. Wtedy $\tau_{m+1} - \tau_m$ ma rozkład geometryczny o parametrze $p = \frac{n_k - m}{n_k}$. $\tau_{\boldsymbol{i},\boldsymbol{j}} = \sum_{m=0}^{n_k-1}(\tau_{m+1} - \tau_m)$, zatem z liniowości wartości oczekiwanej i jej wzoru dla rozkładu geometrycznego:

$$\mathbb{E}(\tau_{ij}) = \mathbb{E}(\sum_{m=0}^{n_k-1}(\tau_{m+1} - \tau_m)) = \sum_{m=0}^{n_k-1}\mathbb{E}(\tau_{m+1} - \tau_m) = \sum_{m=0}^{n_k-1}\frac{n_k}{n_k - m} = n_k\sum_{m=1}^{n_k}\frac{1}{m} \sim n_k\log(n_k)$$

dostajemy więc ostatecznie $d(t) \leq \frac{O(n_k\log n_k)}{t}$. Widać więc, że epsilonowy *mixing time* jest rzędu $\varepsilon\cdot O(n_k\log n_k)$. Korzystając jeszcze z rozważań znanych z problemu kolekcjonera kuponów, można zauważyć, że po $n_k\log n_k$ krokach d spada wykładniczo co kolejne n_k kroków, mianowicie:

$$\begin{aligned} P(\tau_{i,j} > n_k\log n_k + cn_k) &\leq P(\text{pewna ustalona współrzędna nie została odwiedzona}) \\ &= (1 - \frac{1}{n_k})^{n_k\log n_k + cn_k} < e^{-c} \end{aligned}$$

Tak więc $d(n_k \log n_k + cn_k) < e^{-c}$, jak również ostatecznie $t_{mix}(e^{-c}) < n_k \log n_k + cn_k = O(n_k \log n_k)$. Był to oczywiście nie do końca nasz łańcuch – nasz jest bardziej skomplikowany, ale czyniąc pewne heurystyczne założenia możemy coś oszacować i w tym przypadku. Załóżmy więc, że nasz poszukiwany stan *dominuje* łańcuch po współrzędnych w tym sensie, że współrzędne które pokrywają się ze współrzędnymi poszukiwanego stanu: jeśli zostaną podstawione jako kandydat są przyjmowane, natomiast łańcuch 'bardzo niechętnie' je opuszcza. Tak że bardzo długo tam pozostaje, gdy raz już trafi. Efekt ten na początku może być niewidoczny, ale powinien się kaskadowo potęgować wraz z kolejnymi trafieniami. Wtedy możemy przeprowadzić to samo rozumowanie co dla łańcucha 'jednostajnego' tylko pamiętając, że trafienie na współrzędnej musi być trafieniem we współrzędną taką jak w kluczu maksymalnym, stąd prawdopodobieństwa są przemnażane przez $\frac{1}{s}$, więc wynikowa wartość oczekiwana rozkładów geometrycznych będzie przemnożona przez s, tak samo s trzeba wziąć pod uwagę przy późniejszym szacowaniu. Ostatecznie dostajemy przy oznaczeniu τ_{max} dla pierwszego osiągnięcia stanu maksymalnego więc (heurystycznie) zbieżność:

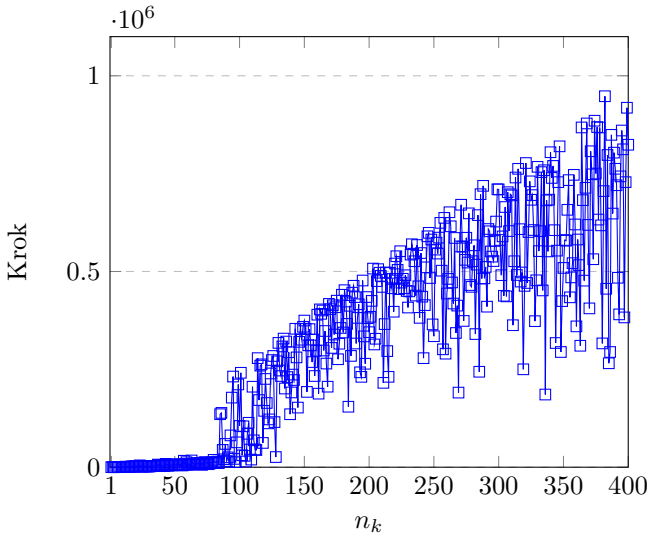
$$P(\tau_{max} > C(n_k \log n_k + cn_k)) < e^{-\frac{c}{s}},$$

czyli

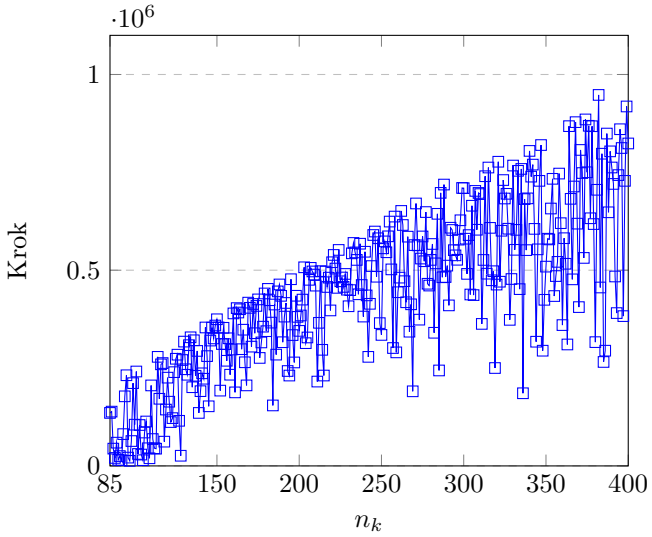
$$P(\tau_{max} > Cs(n_k \log n_k + cn_k) < e^{-c},$$

gdzie C jest pewną stałą związaną zwłaszcza z początkowym 'błądzeniem' łańcucha – zanim zaczął trafiać w kolejne współrzędne – a także z potencjalnymi maksimami lokalnymi. C powinno być *względnie* małe, a przede wszystkim ograniczone z góry, niezależnie od rozmiaru wejścia (10 będzie maksymalnym, jakie przyjmiemy w tej pracy). Jeśliby ta heurystyczna analiza miała pokrycie w rzeczywistości (chociaż w jakiejś części) – pamiętając, że dla tekstu o długości m i klucza długości n_k krok iteracyjny miał złożoność $O(m/n_k)$ – dostaniemy zbieżność (gdzie kolejne przedziały czasowe tej długości niosą wykładniczą poprawę zbieżności) w $O(m \cdot s \cdot \log n_k)$ co oczywiście jest znaczącą poprawą w porównaniu z algorytmem deterministycznym.

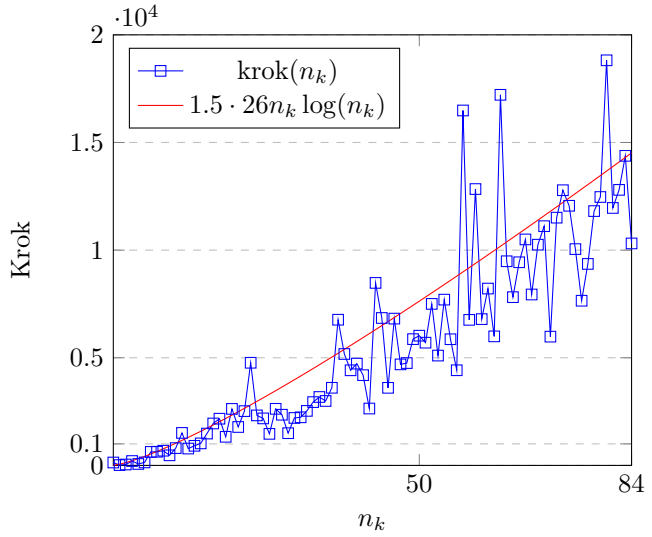
W praktyce rozumowanie to jest poprawne przede wszystkim dla nie największych długości klucza – tam rzeczywiście *dominacja po współrzędnych* ma miejsce, podobnie jak skupienie wszystkich maksimów w małym otoczeniu. Doświadczenia pokazują, że dla wartości klucza, które są już naprawdę duże, opis zbieżności łańcucha staje się bardziej skomplikowany (poza może metodą monogramową, ale ponieważ dla niej algorytm deterministyczny jest szybki, więc nie ma sensu stosować tu wersji MCMC). Aby to zobrazować, przedstawię wyniki testu. Test polegał na zastosowaniu bigramowego algorytmu MCMC do odszyfrowania 1000-znakowego fragmentu tekstu *Madame Bovary* [31] (szyfr Vigenère'a, 26-elementowy alfabet). Ustalamy maksymalną liczbę kroków na $2500n_k$, a wynikiem pojedynczego doświadczenia jest krok, w którym algorytm doszedł do *najlepszego* stanu swojej wędrówki (o największej wadze, nie wiemy, jaki ten stan ma dokładnie związek z *argmaxem*). Chcemy zobaczyć, czy jest jakaś regularność w dochodzeniu do tego stanu. Test przeprowadzamy dla wszystkich dł. klucza od 1 do 400. Klucze dekodujący i startowy wędrówki preparujemy w ten sposób, żeby rzeczywiście różniły się na każdej współrzędnej. Zacznijmy od wykresu zawierającego wszystkie wyniki.



Widać, że wykres wyraźnie dzieli się na dwie części – do ok. 85. kroku wygląda stabilnie (niewielka wariancja) i płasko, po tym czasie wariancja się znacząco zwiększa, a liczba kroków, po której został osiągnięty *najlepszy stan* gwałtownie rośnie – następnie (choć z dużo większą wariancją) rośnie *mniej więcej* liniowo. Zobaczmy osobno część wykresu, która rozpoczyna się od długości 85.

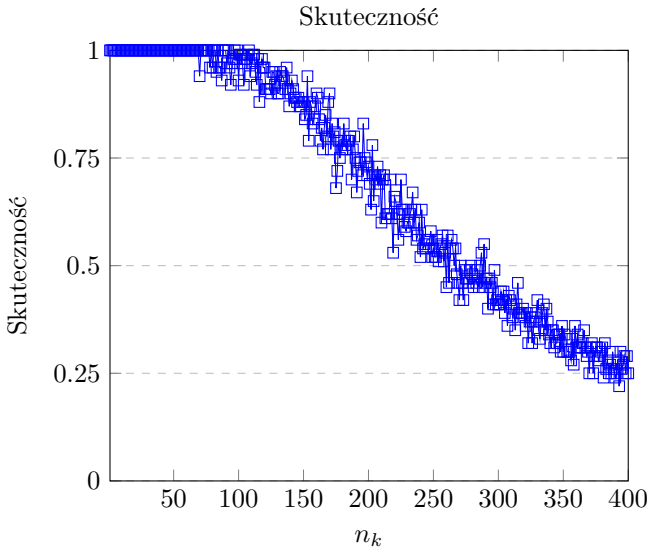


Zobaczmy teraz, jak wygląda wykres do długości 84:



Jak się okazuje, ten fragment jest świetnie przybliżany przez $1.5 \cdot 26 n_k \log n_k$, co oczywiście jest argumentem za poprawnością heurystyki na tym przedziale.

Na koniec zobaczmy, jak się miały uzyskane wyniki do rzeczywistego klucza dekodującego – tj. zobaczmy skuteczność odszyfrowania.



Widzimy, że zależność widoczna na wykresie jest podobna do tych obserwowanych dla bigramowej metody deterministycznej. Teraz, doświadczenie potwierdza, że heurystyka oddaje w dużym stopniu rzeczywistość dla $n_k < 85$ (wtedy też *argmax* jest praktycznie w całości zgodny z rzeczywistym kluczem dekodującym, więc można z dużym prawdopodobieństwem stwierdzić, że został *argmax* osiągnięty w oczekiwanym czasie). Powyżej tej wartości zgodność nie jest tak pewna. Późniejsze doświadczenie pokaże, że – przynajmniej w takim czasie – zbieżność do *argmax* nie zachodzi, raczej jest to zbieżność do pewnego maksimum lokalnego (oczywiście nie zbieżność sensu stricto, a zbieżność w sensie 'w okolicy którego stanu – jako pierwszego – łańcuch zatrzyma się na dłużej'). Jak widać jednak po wykresie skuteczności, język angielski wydaje się posiadać pożądaną cechę – maksima lokalne do pewnej długości są położone *blisko* rzeczywistego klucza dekodującego.
Trudno powiedzieć – bez osobnych doświadczeń – jaka jest natura zbieżności dla tych naprawdę większych n_k. Części wykresu $n_k < 85$ i $n_k \geq 85$ na tyle się różnią , że trudno sobie wyobrazić, żeby pasowało do nich dokładnie to samo rozumowanie. Będzie warto ten temat jeszcze zbadać w przyszłości. Jednak, mimo wyraźnej niedoskonałości, będziemy używać tej czy podobnej heurystyki (różne kombinacje ograniczeń liniowo-logarytmicznych $s \cdot n_k \log n_k$ i liniowych $s \cdot n_k$ na liczbę kroków), sprawdziła się ona bowiem dobrze w wielu testach. Jak się okaże, najważniejsza część heurystyki, co zobaczymy później, wydaje się mieć pokrycie w rzeczywistości. Jest to liniowa zależność zbieżności od s – liczby szyfrów dł. 1. To dokładnie ta własność zadaje największą czasową poprawę w porównaniu z algorytmami deterministycznymi.

4.7 Odgadywanie długości klucza

Jest to problem nieco poboczny względem głównych punktów zainteresowania, jednak może warto go poruszyć. Rozważaliśmy do tej pory szyfrogram zakodowany kluczem o znanej długości – choć nie jest to głównym celem tego rozdziału (najważniejszym jest pokazanie szybkiej metody dekodowania dla *długich* ustalonej długości kluczy), często najważniejszym elementem złamania szyfru typu Vigenère'a jest właśnie odgadnięcie długości klucza. Zostało opracowanych szereg metod dla rozwiązania tego zagadnienia [13], [14], jednak zamierzałem zastosować w jakiś prosty sposób rozwijane tu metody n-gramowe. Pierwszym moim pomysłem było założyć pewne ograniczenie długości klucza – z tego powodu, że po prostu, co pokazały wyniki z jednego z poprzednich podrozdziałów, zbyt duża długość klucza w stosunku do długości tekstu implikuje praktyczną nierozwiązywalność problemu. Następnie myślałem o skonstruowaniu łańcucha od razu dekodującego całość szyfrogramu bez znajomości długości klucza, który poruszałby się po takiej przestrzeni kluczy o ograniczonej długości – pomysłów na macierz generującą było kilka – zawsze częścią macierzy były elementy różniące się z aktualnym stanem na jednej współrzędnej (jak przy ustalonej długości), jednak należało również dodać kandydatów zmieniających długość klucza. Próbowałem robić to na różne sposoby, dodając nową współrzędną na koniec/usuwając współrzędną końcową, dopuszczając dodawanie współrzędnej 'w środek' itd. Nie udało się jednak osiągnąć dobrych wyników, łańcuch miał tendencję do wpadania w maksima lokalne. Możliwe jednak, że da się skonstruować taki łańcuch – pomysłem mogłoby być sprawdzanie na bieżąco, czy zdekodowane słowa istnieją

w języku angielskim, problemem może być jednak znaczące wydłużenie w ten sposób czasu pojedynczego kroku. Choć na ten moment istnienie szybkiego rozwiązania w 'pojedynczej wędrówce' wydaje mi się to mało prawdopodobne, byłoby to najbardziej eleganckie i najbardziej w duchu MCMC rozwiązanie problemu.
Można jednak też praktycznie wprost zastosować metodę n-gramów. Do pewnego progu p dobrze działa metoda monogramów. Tak więc do tego progu próbujemy kolejno dla wszystkich mniejszych lub równych długości tego progu. Jeśli do tego momentu nie udało się uzyskać dobrego rozwiązania (kryterium może być np. liczba/procent poprawnych słów w zdekodowanym tekście) dalej próbujemy metody bigramowej ze startem w monogramowym *argmaksie*. Zwróćmy uwagę, że jeśli chcielibyśmy wykonać dla największej rozważanej długości klucza (powiedzmy q), t kroków, to zgodnie z heurystyką dla klucza u razy krótszego wystarczy $t(\frac{\log \frac{q}{u} + \frac{cq}{u}}{\log q + cq})$ kroków. q powinno być w granicach 1/4 długości tekstu – wtedy jeszcze zdarza się względnie często, że umiemy odszyfrować tekst na poziomie $> 50\%$, dalej metoda zawodzi (co więcej produkuje fałszywie dobre rozwiązania - im więcej dowolności, im dłuższy klucz, tym większą wagę można wyprodukować), więc należy ją w odpowiednim momencie zatrzymać. Długość klucza ostatecznie wybieramy w oparciu o kryterium liczby poprawnych angielskich słów w dekodowanym tekście, po czym dla tej długości ulepszamy rozwiązanie algorytmem bigramowym MCMC z heurystycznie wybraną liczbą kroków. W tym miejscu pokażę jeszcze fragment tekstu (zdekodowany na poziomie 50%), aby uwydatnić, że jest to jeszcze niezły poziom dekodowania, który zazwyczaj możemy 'dodekodować' do końca:
A CLOCE RTOOO AND THOUGIT HA IPE WNTTTT-ALIETY WHAN REAFLD CU HAS NOUGHT FIFOF-ITLIIE OI ALE SOLLOSING MIRSCOR. BUT HE DID OOS WGGEXV INE OOSUTHT FUNTHER. CT BUT YOT INTERETTHUS.
Jeszcze lepiej sprawa wygląda dla 75%:
HE EDGE OF THE LONDED WOOT AND ADYR ITT TISIC INTO NOTHINGNESS? HE WNGIERET WHEEWIR AFULF ALL THERE WAS A MICROPHOND ANDDED SOMPLLERE OLOR. HE AND JULIA HAD SPOKEN OMED IN LEW WHTHTERS, BUR IT WOULD
Widać, że w obu przypadkach tworzą się znajome ciągi, które możemy następnie pouzupełniać. Dla 25% już wiele nie widać:
R PQHONQ, WYSM AO L HKANW OF NISN DMVD ZEZS ONXXQWRGSP TNRBHKH MHA MPWY TO A BTLE. COEZLOV 6 PT BAX WKHORBQD GT YNWT. MHA PYIEHCED XYTSHKO ZKH JOGE. UAV ZHF ZUFK
Ogólnie, w każdym wypadku, jakiekolwiek zwiększenie poprawności dekodowania, niesie ze sobą dużą wartość. Jeśli jesteśmy poniżej 50%, to zbliżenie nas do 50% daje szanse na zdekodowanie tekstu w ogóle. Jeśli przeskok jest z np. około 50% do 80-100%, to pozwala nam to bardzo przyspieszyć proces. W każdym wypadku taka poprawa jest bardzo pożądana – stąd użycie $(n > 1)$-gramów jest tak istotne.

Wracając do tematu nieznanej długości klucza, biorąc pod uwagę heurystyczną optymalizację uzyskamy złożoność rzędu $p \cdot s \cdot m$, gdzie m długość tekstu, jeśli udało się osiągnąć wymaganą poprawność szyfrowania dla $n_k < p$ a w przeciwnym wypadku, zakładając, że t jest zgodne z heurystyką, dostajemy $O(s \cdot m \cdot \log^2 q)$, bowiem dzielimy kolejne liczby kroków przez stosunek długości odpowiadającego klucza do maksymalnego rozważanego (powstaje liczba harmoniczna, a nawet coś mniejszego, bo dzielimy jeszcze dodatkowo przez logarytm, ale takie oszacowanie nam wystarczy). Na koniec wypada zauważyć, że przy dostępie do pól słownika w stałym czasie, a taki zakładamy, sprawdzenie liczby poprawnych słów daje złożonośc $O(m)$, więc nie zwiększa ustalonej złożoności.

4.8 Wyniki, wnioski i podsumowanie rozdziału

Na początek tego rozdziału uwaga. Do tej pory w pracy czyniliśmy założenie, że dekodowane teksty są rzeczywiście fragmentami tekstów w poprawnym języku angielskim. Oznaczało to, że poza – możliwe – pierwszym i ostatnim, wszystkie *słowa* pochodzą z języka angielskiego, więc i wszystkie *n-gramy* są dla tego języka naturalne. Czasami przy badaniu szyfru Vigenère'a, usuwa się po prostu niekodowane znaki w oryginalnym tekście, m.in. spacje, co powoduje, że zamiast wielu, dostajemy jedno bardzo długie słowo, w którym występują 'oszukane' n-gramy, jeśli $n > 1$. 'Oszukane', ponieważ są to n-gramy powstałe z liter wziętych z końcówki jednego słowa i początku następnego, czyli 'niesłownikowe'. Można się zastanawiać, co stanie się, jeśli do odszyfrowania

dostaniemy taki właśnie tekst, na ile to osłabi moc metody bigramowej (na metodę monogramową, łatwo zauważyć, nie ma to żadnego wpływu). Jak się okazuje, pewien wpływ jest, jednak nie jest on *bardzo* wielki – metoda bigramowa cały czas ma dużą moc. Poniżej przykładowe wyniki dla losowo wygenerowanego 814-znakowego tekstu (powstałego przez usunięcie niekodowanych znaków z losowego 1000-znakowego tekstu) z *Roku 1984*. Użyłem tych samych co wcześniej słowników log-częstości wygenerowanych na bazie *Wojny i pokoju*, *Dumy i uprzedzenia* oraz *Olivera Twista* z projektu Gutenberg [31]:

	SKUTECZNOŚĆ	
DŁUGOŚĆ KLUCZA	METODA MONOGRAMOWA	METODA BIGRAMOWA
20	1.0	1.0
40	1.0	1.0
60	0.95	1.0
80	0.86	0.97
100	0.75	0.96
120	0.66	0.9
140	0.52	0.83
160	0.48	0.78
180	0.47	0.72
200	0.39	0.56
220	0.35	0.50
240	0.35	0.52
260	0.34	0.45
280	0.30	0.35
300	0.25	0.33
320	0.24	0.34
340	0.23	0.31
360	0.27	0.30
380	0.22	0.33
400	0.23	0.23

Tabela 9: Skuteczność deterministycznych metod n-gramowych na 814-znakowym fragmencie *Roku 1984* pozbawionym przerw między słowami zaszyfrowanym klasycznym szyfrem Vigenère'a

Widać więc pewne niewielkie osłabienie metody bigramowej, jednak cały czas zadaje ona dużą poprawę w porównaniu z monogramową – wyniki jak powyższy są powtarzalne, często metoda bigramowa radzi sobie nawet jeszcze lepiej. Można oczywiście problemowi dekodowania tego typu (i potencjalnym modyfikacjom redukującym efekt 'oszukanych' n-gramów) poświęcić więcej czasu w osobnej pracy, widać jednak, że metoda bigramowa (czy generalnie $n > 1$-gramowa) sprawdzi się i tu.

Następnym pobocznym punktem, który chciałbym poruszyć, jest możliwość, że szyfr nakładamy również na znaki niekodowane. O co chodzi? Przypomnijmy sobie przykład pokazujący ideę Vigenère'a z jednego z poprzednich podrozdziałów, kodowanie następowało wg poniższej tabeli:

M	O	N	T	E		C	A	R	L	O		M	A	R	K	O	V		C	H	A	I	N	Z
B	C	B	C	B		C	B	C	B	C		B	C	B	C	B	C		B	C	B	C	B	C	

Ale można by też podkładać znaki szyfru pod niekodowane spacje (oczywiście wynikiem kodowania nadal byłaby spacja):

M	O	N	T	E		C	A	R	L	O		M	A	R	K	O	V		C	H	A	I	N	Z
B	C	B	C	B	C	B	C	B	C	B	C	B	C	B	C	B	C	B	C	B	C	B	C	B

Algorytm zadziała wtedy praktycznie bez różnicy, jeśli tylko zmodyfikujemy go tak, by brał pod uwagę inny sposób podkładania szyfru. Gdy nie wiemy, który sposób został użyty należy przeprowadzić dwie próby ataku (na opcję nr 1 i opcję nr 2). Stąd też szybkość ma znaczenie: im

szybciej jesteśmy w stanie przeprowadzić próbę ataku na pojedynczy typ szyfru, tym więcej typów będziemy w stanie zaatakować w rozsądnym czasie.

Były to poboczne rozważania, wracamy do wersji kodowania, którą zajmujemy się w tej pracy. Będziemy pracować głównie na trzech alfabetach:
(1) klasycznym ABCDEFGHIJKLMNOPQRSTUVWXYZ
(2) rozszerzonym o spację alfabecie 'alpha' z [17] – jest tu opisany nowy algorytm szyfrowania oparty na schemacie Vigenère'a – z punktu widzenia naszej pracy nie różni się niczym od algorytmu Vigenère'a tak jak my go rozumiemy (przypomnijmy, że definiowaliśmy go na dowolnym alfabecie). Dodatkowym elementem szyfrowania jest tu jeszcze permutacja (szyfr podstawieniowy monoalfabetyczny) zastosowana po schemacie Vigenère'a – ale ponieważ permutacja (podstawienie) jest znane, więc wystarczy najpierw zastosować na szyfrogramie permutację odwrotną, a następnie nasze zwykłe algorytmy. Permutacja ta jest utrudnieniem tylko gdy nie wiemy, że został użyty opisany szyfr, w przeciwnym wypadku nie ma na nic wpływu. Rozszerzenie o spację zapewnia, że zakodowane będą wszystkie znaki (czyścimy wcześniej tekst ze znaków nowej linii tabulatorów – białych znaków innych niż spacja). Alfabet ma postać:

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
‘ !@$%^*()_-=+[]|;:«>,.?/ +spacja

(3) alfabet jak w (2), z tym że bez spacji (czyli oryginalny alfabet z [17]).

Przypomnijmy jeszcze raz, że używamy słowników log-częstości wygenerowanych na bazie *Wojny i pokoju*, *Dumy i uprzedzenia* oraz *Olivera Twista*. Słowniki są różne dla dwu alfabetów, dla (1) bowiem oczywiście małe litery musiały być przekształcone na wielkie.

Zanim przejdziemy do głównego tematu pracy: łamania długich w stosunku do długości zaszyfrowanego tekstu szyfrów, zaprezentuję pokrótce wyniki dekodowania szyfrogramu (zakodowanego zwykłym szyfrem Vigenère'a, gdzie długość klucza jest nieznana, ale mniejsza od $q = 1/4$ długości tekstu). Losujemy 1000-znakowy fragment z *Roku 1984* (oczywiście z małymi literami przekształconymi na wielkie) i szyfrujemy kolejno kluczami o długości 10, 50, 100, 150, 200, 240. Następnie próbujemy rozszyfrować tekst zgodnie z opisaną w poprzednim podrozdziale metodą, nie znając długości klucza. Przyjmujemy próg $p = 1/8$, dla którego wstępne sprawdzenie wykonujemy jedynie algorytmem monogramowym, dalej $q = 1/4$, a następnie, że wykonujemy $\frac{1}{4} \cdot 26 \cdot (n_k \log n_k + 1 \cdot n_k)$ kroków w celu wstępnego sprawdzenia dla długości większych niż $1/8$, czyli przyjmujemy $C = \frac{1}{4}$ i $c = 1$ dla oznaczeń z rozdziału o *mixing time*, natomiast $s = 26$ – jest to długość angielskiego alfabetu. Jeśli udało się uzyskać wstępnym dekodowaniem powyżej 50% słów istniejących w języku angielskim (jako zbioru słów angielskich używamy [35]), uznajemy, że odgadliśmy poprawną długość i nie sprawdzamy większych. Nominalnie, odgadnięta długość klucza jest tą, dla której używając metod n-gramowych dostaliśmy najwięcej poprawnych słów. Następnie dla wybranej długości dodatkowo ulepszamy rozwiązanie metodą bigramową MCMC dla $\frac{1}{4} \cdot 26 \cdot (n_k \log n_k + 10 \cdot n_k)$ kroków, czyli przyjmujemy $c = 10$, $C = \frac{1}{4}$. Ogólnie złożoność $O(s \cdot m \cdot \log^2 q)$ dostaje stałą $\frac{1}{4}$ (dodatkowe wywołanie algorytmu bigramowego nie zwiększa istotnie złożoności czasowej). Otrzymujemy wyniki jak poniżej:

RZECZYWISTA DŁUGOŚĆ	ODGADNIĘTA DŁUGOŚĆ	SKUTECZNOŚĆ
10	10	1.0
50	50	1.0
100	100	0.95
150	150	0.86
200	200	0.65
240	240	0.5

Tabela 10: 1000-znakowy fragment *Roku 1984*, klasyczny szyfr Vigenère'a z 26-znakowym alfabetem o nieznanej długości klucza

Są to naprawdę niezłe wyniki. Stała C jest dobrana raczej losowo, a mimo to dostajemy dobry poziom dekodowania (w przyzwoitym czasie kilku minut). Ogólnie różne wartości zostały tu dobrane *a priori* i można je zoptymalizować, można też użyć innych *gramów*. Jest to jedynie poboczny wynik tej pracy i tylko ocieram się o ten temat, natomiast widać, że za pomocą samych metod n-gramowych można wiele w dekodowaniu osiągnąć. Widać tutaj znowu jak ważna jest szybkość dekodowania: użycie deterministycznej metody bigramowej, nie pozwoliłoby nam uzyskać wyniku w sensownym czasie. Użycie samej metody monogramowej do odgadnięcia długości regularnie zawodziło dla 240. Stąd metody MCMC okazują się w tym miejscu nieodzowne.

Przechodzę teraz do głównej części testowania, czyli testowania różnych szyfrów przy założeniu znajomości długości klucza. Testy przeprowadzimy na wygenerowanych losowych 1000-znakowych fragmentach *Roku 1984*, *Madame Bovary* Gustave'a Flauberta [31] oraz artykułu *Ice hockey* [36] z angielskiej Wikipedii. Ostatecznie decydujemy się na następujące podejście: deterministycznie wybieramy stan maksymalizujący log-wagę monogramową i z niego zaczynamy bigramową wędrówkę MCMC przez liczbę kroków ustaloną *a priori* na wielokrotność $s \cdot n_k \log n_k$, $s \cdot n_k$ lub kombinację dwóch. Zaczniemy od stałych $C = 5$ i $c = 5$ zastosowanych do wzoru z heurystycznych rozważań (a więc $5s(n_k \log n_k + 5n_k)$), następnie będzie można ograniczenie to zaktualizować. Dekodowanie bigramowe jest wynikiem pierwszej wędrówki (*bigram skuteczność* to skuteczność po zakończeniu tej wędrówki). Potem wędrówkę jeszcze przedłużamy o spacer na trigramach, startując w *najlepszym* stanie wędrówki bigramowej (wędrówka trigramowa potrwa początkowo 1/4 liczby kroków na z wędrówki bigramowej) i dostajemy informację o skuteczności (*trigram skuteczność* to skuteczność po zakończeniu wędrówki trigramowej). Testy przeprowadzamy najpierw dla klasycznego szyfru Vigenère'a i długości klucza zmieniających się od 40 do 400 (co 40). Dla alfabetu nr 1 (klasycznego) z oczywistych powodów małe litery są przekształcane na wielkie.
Korzystając z tego, że dla 26-elementowego alfabetu i szyfru Vigenère'a (jest to prawdą także dla szyfru z autokluczem) jesteśmy jeszcze w stanie wyliczyć *argmax* w sensownym czasie, to dla pierwszego testu (*Rok 1984* i klasyczny szyfr Vigenère'a) zbadamy zbieżność wędrówki bigramowej do *argmax*, tj. będą podane kroki, w których MCMC osiągnął daną zgodność z bigramowym kluczem *argmaxowym*, przy czym jeśli *argmax* zostaje osiągnięty, to kończymy wędrówkę. Ponadto wskażemy czasy działania obu algorytmów, aby pokazać jak istotna jest różnica w ich złożoności czasowej.

Zaczynamy od przetestowania 1000-znakowego urywku z *Roku 1984* dla szyfru Vigenère'a, dostając najpierw wyniki jak poniżej dla algorytmu deterministycznego:

DŁUGOŚĆ KLUCZA	CZAS	SKUTECZNOŚĆ
40	88.7s	1.0
80	87.0s	1.0
120	90.1s	0.98
160	92.8s	0.96
200	90.0s	0.82
240	91.3s	0.68
280	90.5s	0.53
320	91.5s	0.48
360	91.4s	0.3
400	92.0s	0.35

Tabela 11: *Rok 1984* klasyczny szyfr Vigenère'a (alfabet (1)), deterministyczny algorytm bigramowy

Natomiast dla algorytmu MCMC:

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM ZBIEŻNOŚĆ	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.01s	0.925	0.39s	0.95: 152 1.0: 1021 max kroków: 45182	1.0	4.53s	1.0

80	0.01s	0.725	0.94s	0.75: 238 0.8: 954 0.85: 1518 0.9: 1939 0.95: 2360 1.0: 6473 max kroków: 97573	1.0	4.73s	1.0
120	0.01s	0.47	15.2s	0.5: 1067 0.55: 3330 0.6: 3689 0.65: 4736 0.7: 5927 0.75: 8135 0.8: 10909 0.85: 22794 0.9: 112101 0.95: - 1.0: - max kroków: 152684	0.90	5.03s	0.92
160	0.01s	0.46875	16.0s	0.5: 1471 0.55: 3573 0.6: 6947 0.65: 8352 0.7: 13283 0.75: 14948 0.8: 30025 0.85: 44571 0.9: 68164 0.95: - 1.0: - max kroków: 209563	0.90	5.19s	0.96
200	0.01s	0.34	16.6s	0.45: 566 0.5: 2548 0.55: 4026 0.6: 6480 0.65: 17549 0.7: 39572 0.75: 63315 0.8: 115273 0.85: - 0.9: - 0.95: - 1.0: - max kroków: 267756	0.74	5.44s	0.83
240	0.02s	0.32	17.2s	0.45: 7368 0.5: 9315 0.55: 31877 0.6: 86299 0.65: 224267 0.7: - 0.75: - 0.8: - 0.85: - 0.9: - 0.95: - 1.0: - max kroków: 326995	0.54	5.58s	0.68
280	0.01s	0.28	17.9s	0.5: 7355 0.55: 11911 0.6: 28644 0.65: 76892 0.7: 205922 0.75: - 0.8: - 0.85: - 0.9: - 0.95: - 1.0: - max kroków: 387106	0.50	5.73s	0.63
320	0.01s	0.26	18.9s	0.4: 10821 0.45: 60024 0.5: 111411 0.55: - 0.6: - 0.65: - 0.7: - 0.75: - 0.8: - 0.85: - 0.9: - 0.95: - 1.0: - max kroków: 447962	0.36	5.93s	0.47
360	0.02s	0.20	19.1s	0.4: 39469 0.45: 107998 0.5: 221492 0.55: - 0.6: - 0.65: - 0.7: - 0.75: - 0.8: - 0.85: - 0.9: - 0.95: - 1.0: - max kroków: 509469	0.28	6.06s	0.37
400	0.02s	0.21	19.6s	0.4: 28323 0.45: 47567 0.5: 200121 0.55: - 0.6: - 0.65: - 0.7: - 0.75: - 0.8: - 0.85: - 0.9: - 0.95: - 1.0: - max kroków: 571556	0.29	6.23s	0.34

Tabela 12: *Rok 1984*, 1000 znaków, klasyczny szyfr Vigenère'a (alfabet (1), skuteczność metod 1,2,3-gramowych MCMC. Zbieżność metody bigramowej MCMC

Zauważmy, że na pewno łańcuch bigramowy od pewnej długości klucza nie zbiega prosto do *argmax*, a raczej do pewnego maksimum lokalnego – od pewnej długości klucza pojawiają się maksima lokalne (w nie aż tak dalekim otoczeniu rzeczywistego *argmaxu*, bo w dużej mierze są z nim zgodne). Widać, jak już było wspomniane, że język angielski zdaje się mieć właściwość, że jeszcze dla całkiem długich kluczy, maksima lokalne mają sporą zgodność z rzeczywistością (prawdziwym kluczem dekodującym). Dokładniejsze teoretyczne przeanalizowanie zachowania łańcucha dla sytuacji, gdy klucze zaczynają być bardzo długie, pozostaje zadaniem do zrealizowania. My jednak w inny sposób skorzystajmy z wyniku powyżej. (1) zwróćmy uwagę, że użycie wędrówki trigramowej jako kontynuacji bigramowej dało świetne rezultaty (często wyniki lepsze niż bigramowy algorytm deterministyczny). (2) Jeśliby używać takich ograniczeń jak w tym teście na liczbę kroków, zysk czasowy – jeden z ważniejszych celów – byłby stosunkowo niewielki (choć oczywiście znacząco by wzrósł dla większych s), więc musimy tę liczbę kroków zmniejszyć, (3) metoda bigramowa 'zatrzymuje się' zwykle wcześniej niż my ją zatrzymujemy. Odtąd przyjmujemy więc uproszczoną wartość $n_k \cdot s \cdot 10$, przy czym tyle samo kroków wykonujemy najpierw w wędrówce bi-, a potem trigramowej. Tak kontynuujemy testy już dla wszystkich przypadków.

Dla losowego 1000-znakowego fragmentu *Madame Bovary* (dekodowanie tekstu to jednocześnie świetna okazja, żeby obcować z klasyczną literaturą) dla klasycznego szyfrowania Vigenère'a przy alfabecie (1) dostajemy :

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.01s	1.0	2.76s	1.0	3.66s	1.0
80	0.01s	0.83	2.82s	1.0	3.71s	1.0
120	0.01s	0.6	2.91s	0.89	3.96s	0.98
160	0.01s	0.46	2.99s	0.83	3.88s	0.91
200	0.01s	0.44	3.02s	0.66	3.96s	0.76
240	0.01s	0.35	3.08s	0.62	3.96s	0.70
280	0.01s	0.28	3.17s	0.46	4.07s	0.62
320	0.01s	0.26	3.19s	0.35	4.10s	0.41
360	0.01s	0.27	3.29s	0.30	4.22s	0.34
400	0.02s	0.25	3.36s	0.26	4.28s	0.3

Tabela 13: *Madame Bovary* dł. 1000, klasyczny szyfr Vigenère'a (alfabet (1))

Jak widzimy wyniki są jak najbardziej zadowalające i co ważne – szybkie (w sumie mniej niż 8s). Małe różnice w czasie działania algorytmu (teoretycznie dla każdej długości klucza, którą testujemy złożoność jest ta sama) wynikają ze specyfiki zbieżności. W początkowych przypadkach łańcuch zbiega bardzo szybko, więc dalej przejścia do innych stanów są bardzo rzadkie. Przejścia do innych stanów wiążą się z m.in. z update'em dekodowania czy słownika częstości *gramów* w dekodowaniu, co kosztuje. Stąd krótszy czas wykonania dla małych kluczy, przejścia bowiem są tam rzadsze.
Wyniki dla *Ice hockey* prezentują się natomiast jak poniżej:

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.02s	0.97	2.77s	1.0	3.69s	1.0
80	0.01s	0.81	2.87s	1.0	3.76s	1.0
120	0.01s	0.6	2.94s	0.98	3.93s	1.0
160	0.01s	0.51	2.99s	0.78	3.90s	0.95
200	0.01s	0.44	3.19s	0.67	4.01s	0.87

240	0.01s	0.38	3.10s	0.57	4.01s	0.7
280	0.01s	0.31	3.19s	0.47	4.10s	0.52
320	0.01s	0.27	3.27s	0.38	4.20s	0.45
360	0.02s	0.25	3.35s	0.36	4.29s	0.46
400	0.02s	0.24	3.38s	0.27	4.33s	0.37

Tabela 14: *Ice hockey* dł. 1000, klasyczny szyfr Vigenère'a (alfabet (1))

Teraz zaprezentujmy ten sam algorytm dla 1000-znakowego fragmentu z każdej testowanej pozycji dla szyfru z autokluczem (nie ma tu wielkiej różnicy w działaniu, jednak warto to przynajmniej zaprezentować).

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.01s	0.97	2.80s	1.0	3.71s	1.0
80	0.01s	0.82	2.88s	1.0	3.84s	1.0
120	0.01s	0.56	2.95s	0.9	3.84s	0.95
160	0.01s	0.51	3.01s	0.83	3.91s	0.97
200	0.01s	0.38	3.10s	0.69	4.36s	0.83
240	0.02s	0.34	3.19s	0.55	4.06s	0.69
280	0.02s	0.27	3.50s	0.46	5.34s	0.54
320	0.02s	0.25	4.05s	0.36	4.48s	0.48
360	0.02s	0.19	3.71s	0.28	4.49s	0.34
400	0.02s	0.16	3.60s	0.22	4.87s	0.29

Tabela 15: Klasyczny alfabet (1), szyfr z autokluczem *Rok 1984* 1000 znaków

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.01s	0.97	2.81s	1.0	3.80s	1.0
80	0.01s	0.78	2.89s	1.0	3.80s	1.0
120	0.01s	0.6	2.98s	0.9	3.85s	0.97
160	0.01s	0.46	3.01s	0.88	3.95s	0.97
200	0.01s	0.36	3.10s	0.65	4.02s	0.76
240	0.02s	0.34	3.14s	0.52	4.13s	0.63
280	0.02s	0.31	3.24s	0.45	4.15s	0.56
320	0.02s	0.21	3.30s	0.35	4.20s	0.43
360	0.02s	0.20	3.37s	0.27	4.32s	0.30
400	0.02s	0.18	3.42s	0.29	4.36s	0.29

Tabela 16: Klasyczny alfabet (1), szyfr z autokluczem *Madame Bovary* 1000 znaków

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.01s	0.95	3.10s	1.0	4.11s	1.0
80	0.01s	0.68	4.05s	0.93	3.96s	0.95
120	0.01s	0.52	3.06s	0.87	4.04s	0.91
160	0.02s	0.33	3.44s	0.66	4.34s	0.83
200	0.01s	0.32	4.11s	0.55	6.17s	0.66
240	0.04s	0.28	3.42s	0.39	4.21s	0.52

280	0.02s	0.27	3.54s	0.37	4.35s	0.45
320	0.02s	0.17	3.41s	0.28	4.29s	0.31
360	0.02s	0.18	3.65s	0.23	4.43s	0.28
400	0.02s	0.15	3.45s	0.2	4.41s	0.26

Tabela 17: Klasyczny alfabet (1), szyfr z autokluczem *Ice hockey* 1000 znaków

Widzimy, że szyfr z autokluczem – jak można było się domyślać – nie różni się z punktu widzenia zautomatyzowanego deszyfrowania od klasycznego szyfru Vigenère'a, dalej nie będziemy się więc nim już osobno zajmować. Jak widać metoda MCMC jest dla niego skuteczna.
Zobaczmy teraz wyniki dla wersji *rozszerzonej/afinicznej* szyfru Vigenère'a: s zmienia się tu na $26 \cdot \phi(26) = 26 \cdot 12 = 312$. Stąd deterministyczny algorytm bigramowy wydłużyłby swoje działanie $12^3 = 1728$ razy. Co oznacza, biorąc pod uwagę wcześniejsze wyniki, 1,8 dnia oczekiwania na rozwiązanie. Traci więc on jakiekolwiek praktyczne zastosowanie. Możemy natomiast znowu zastosować podejście MCMC.

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.20s	0.77	34.3s	1.0	45.3s	1.0
80	0.21s	0.36	35.6s	0.78	46.1s	0.85
120	0.22s	0.13	36.5s	0.43	50.4s	0.55
160	0.22s	0.09	39.1s	0.21	48.2s	0.26
200	0.23s	0.11	38.4s	0.16	49.4s	0.21
240	0.24s	0.06	39.6s	0.06	50.1s	0.10
280	0.24s	0.03	43.5s	0.04	52.0s	0.08
320	0.25s	0.02	42.0s	0.05	53.0s	0.08
360	0.26s	0.03	45.5s	0.03	52.9s	0.05
400	0.26s	0.01	43.7s	0.02	54.4s	0.04

Tabela 18: Klasyczny alfabet (1), szyfr rozszerzony/afiniczny *Rok 1984* 1000 znaków

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.19s	0.67	36.2s	1.0	50.5s	1.0
80	0.20s	0.27	41.0s	0.61	48.2s	0.8
120	0.21s	0.18	41.2s	0.31	51.6s	0.46
160	0.21s	0.11	41.0s	0.26	54.7s	0.28
200	0.23s	0.06	41.7s	0.11	52.7s	0.15
240	0.24s	0.06	43.8s	0.07	59.3s	0.1
280	0.25s	0.03	45.6s	0.05	55.7s	0.06
320	0.40s	0.05	45.1s	0.03	60.5s	0.04
360	0.40s	0.03	49.7s	0.04	62.5s	0.04
400	0.27s	0.02	49.2s	0.02	64.5s	0.03

Tabela 19: Klasyczny alfabet (1), szyfr rozszerzony/afiniczny *Ice hockey* 1000 znaków

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.21s	0.85	37.3s	1.0	48.9s	1.0
80	0.23s	0.33	40.3s	0.73	55.7s	0.88
120	0.26s	0.19	40.7s	0.53	55.2s	0.66
160	0.22s	0.07	43.1s	0.22	58.4s	0.3

200	0.25s	0.09	43.6s	0.08	56.5s	0.11
240	0.29s	0.03	44.5s	0.06	57.4s	0.09
280	0.39s	0.03	45.7s	0.03	61.9s	0.11
320	0.29s	0.04	46.6s	0.04	61.1s	0.07
360	0.31s	0.03	48.8s	0.02	56.7s	0.06
400	0.27s	0.02	45.3s	0.01	60.1s	0.02

Tabela 20: Klasyczny alfabet (1), szyfr rozszerzony/afiniczny *Madame Bovary* 1000 znaków

Trzeba wprost powiedzieć, że działanie nie jest już tak dobre jak dla *przesuwnych* szyfrów Vigenère'a. Albo może powiedzmy inaczej, na początku jest równie dobre, ale szybciej przestaje takie być. Jednak cały czas widzimy, że jest to podejście niezwykle użyteczne. Zbieżność w istocie jest liniowa ze względu na s, widzimy, że tam gdzie był w stanie, algorytm zbiegł w oczekiwanym czasie. Jednak dla bardziej skomplikowanych podstawień na współrzędnych zwyczajnie zmniejsza się moc metody *n-gramowej*. Możemy myśleć sobie, że coraz bardziej zmierzamy do tego, że na współrzędnych mamy dowolne permutacje alfabetu – w takiej, granicznej, sytuacji, przy już nie tak dużej jak wcześniej długości klucza, szyfr zbliża się do szyfru nie do złamania (permutacje dają dużo większe możliwości manipulacji tekstem niż przesunięcia, stąd jesteśmy w stanie wygenerować bardzo prawdopodobną, acz kompletnie niezgodną z zaszyfrowaną sekwencję znaków). Zwróćmy jednak przede wszystkim uwagę, że użycie metod MCMC w powyższych przypadkach dla pewnych długości szyfru zwiększa w porównaniu z metodą monogramową skuteczność nawet 4-5-krotnie. Co oznacza, że z poziomu zupełnej niezdolności odszyfrowania, przechodzimy na poziom, na którym wiadomość odszyfrujemy w zasadzie na pewno.

Przeprowadźmy jeszcze na koniec testy dla szyfru Vigenère'a z rozszerzonym alfabetem z dodatkową spacją (2) i (3) bez dodatkowej spacji. Zacznijmy od (2): dla *Madame Bovary*

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.06s	1.0	12.2s	1.0	16.5s	1.0
80	0.06s	0.96	13.2s	1.0	16.9s	1.0
120	0.07s	0.91	12.9s	1.0	17.2s	1.0
160	0.07s	0.81	13.3s	0.97	17.3s	0.98
200	0.07s	0.76	13.3s	0.98	17.5s	0.99
240	0.07s	0.64	13.4s	0.88	17.6s	0.96
280	0.07s	0.55	13.7s	0.73	18.0s	0.78
320	0.08s	0.56	13.9s	0.70	18.4s	0.78
360	0.08s	0.48	14.2s	0.65	18.4s	0.72
400	0.08s	0.42	14.3s	0.50	18.8s	0.57

Tabela 21: Alfabet (2), szyfr Vigenère'a, 1000 znaków, *Madame Bovary*

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.07s	1.0	12.2s	1.0	16.5s	1.0
80	0.06s	0.98	12.6s	1.0	16.9s	1.0
120	0.07s	0.9	12.8s	0.98	17.4s	1.0
160	0.07s	0.81	13.1s	0.99	17.4s	1.0
200	0.07s	0.75	13.4s	0.96	18.1s	0.98
240	0.10s	0.65	15.4s	0.87	20.6s	0.95
280	0.09s	0.57	19.3s	0.78	19.6s	0.86
320	0.08s	0.53	14.1s	0.77	20.4s	0.86
360	0.09s	0.48	17.3s	0.68	20.9s	0.77
400	0.09s	0.42	17.5s	0.59	18.8s	0.66

Tabela 22: Alfabet (2), szyfr Vigenère'a, 1000 znaków, *1984*

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.06s	1.0	12.5s	1.0	16.8s	1.0
80	0.06s	0.97	12.6s	1.0	16.9s	1.0
120	0.07s	0.85	12.9s	0.98	17.1s	0.98
160	0.07s	0.81	13.4s	0.97	17.3s	0.98
200	0.07s	0.65	13.3s	0.9	17.8s	0.95
240	0.07s	0.6	14.4s	0.86	18.3s	0.93
280	0.08s	0.55	14.2s	0.74	18.9s	0.83
320	0.08s	0.49	14.5s	0.64	18.2s	0.74
360	0.08s	0.42	14.2s	0.63	18.5s	0.72
400	0.08s	0.37	14.3s	0.55	18.6s	0.63

Tabela 23: Alfabet (2), szyfr Vigenère'a, 1000 znaków, *Ice hockey*

Zwraca tu uwagę bardzo (!) wysoka skuteczność już metody monogramowej, choć teoretycznie szyfr wydawałby się trudniejszy, skoro operuje na większej liczbie znaków. Ma to jednak swoje proste uzasadnienie – jest to spacja (oraz dodatkowy powód, o którym powiem zaraz przy okazji alfabetu (3)). W przeciwieństwie do liter alfabetu, w których częstościach występują regularnie fluktuacje, spacja jest niekwestionową 'królową', jeśli idzie o częstość. Metoda monogramowa 'odgaduje' więc jak zakodowana jest spacja, co pozwala zdekodować jej bardzo często poprawnie wiele monoalfabetycznych części szyfrogramu. Można zobaczyć, że podobnie bardzo dobre wyniki otrzymujemy, gdy do alfabetu (1) również dodamy na koniec spację:

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.01s	0.97	3.30s	1.0	4.42s	1.0
80	0.01s	0.93	3.38s	1.0	4.48s	1.0
120	0.01s	0.80	3.44s	1.0	4.54s	1.0
160	0.02s	0.7	3.53s	0.94	4.58s	0.96
200	0.02s	0.54	3.58s	0.79	4.69s	0.88
240	0.02s	0.51	3.68s	0.77	4.82s	0.86
280	0.02s	0.46	3.72s	0.62	4.83s	0.68
320	0.02s	0.36	3.82s	0.53	4.93s	0.63
360	0.02s	0.35	3.88s	0.43	4.94s	0.51
400	0.02s	0.33	3.88s	0.39	4.98s	0.40

Tabela 24: *Madame Bovary*, 1000 znaków, klasyczny alfabet (1) z dodaną spacją na koniec – szyfr Vigenère'a

Wyniki także ulegają znaczącej poprawie, jednak są trochę gorsze niż dla alfabetu (2). By zobaczyć dlaczego, spójrzmy na wyniki dla alfabetu (3):
1984

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.06s	1.0	10.2s	1.0	13.7s	1.0
80	0.05s	0.95	10.6s	1.0	14.0s	1.0
120	0.05s	0.84	10.8s	0.99	14.2s	1.0
160	0.06s	0.68	11.1s	0.95	14.4s	0.98
200	0.06s	0.60	11.4s	0.92	14.7s	0.96
240	0.06s	0.52	11.5s	0.80	14.9s	0.91
280	0.06s	0.42	11.8s	0.67	15.1s	0.79

320	0.06s	0.38	12.1s	0.56	15.6s	0.69
360	0.07s	0.32	12.1s	0.52	15.6s	0.61
400	0.07s	0.30	12.3s	0.44	15.9s	0.55

Tabela 25: Alfabet (3), szyfr Vigenère'a, 1000 znaków, *Rok 1984*

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.05s	1.0	10.7s	1.0	15.2s	1.0
80	0.05s	0.98	10.8s	1.0	14.3s	1.0
120	0.05s	0.78	11.4s	0.95	14.5s	1.0
160	0.06s	0.67	11.2s	0.98	14.8s	1.0
200	0.06s	0.61	11.4s	0.93	15.2s	0.95
240	0.06s	0.51	11.7s	0.78	15.4s	0.9
280	0.07s	0.43	12.0s	0.64	15.6s	0.78
320	0.06s	0.37	12.4s	0.56	15.7s	0.67
360	0.07s	0.30	12.2s	0.50	16.1s	0.63
400	0.07s	0.28	12.5s	0.40	16.3s	0.51

Tabela 26: Alfabet (3), szyfr Vigenère'a, 1000 znaków, *Madame Bovary*

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.05s	0.97	11.2s	1.0	14.8s	1.0
80	0.05s	0.82	11.8s	1.0	15.6s	1.0
120	0.06s	0.68	11.5s	0.90	15.1s	0.96
160	0.06s	0.6	11.8s	0.86	19.1s	0.95
200	0.07s	0.43	13.2s	0.76	17.4s	0.84
240	0.06s	0.37	12.2s	0.62	15.7s	0.75
280	0.06s	0.35	12.5s	0.58	16.3s	0.68
320	0.07s	0.29	12.5s	0.49	16.2s	0.6
360	0.07s	0.27	12.6s	0.44	16.5s	0.52
400	0.07s	0.25	13.5s	0.34	16.7s	0.36

Tabela 27: Alfabet (3), szyfr Vigenère'a, 1000 znaków, *Ice hockey*

Wyniki są trochę gorsze niż dla (2), ale również bardzo dobre – lepsze niż w klasycznym szyfrowaniu Vigenère'a. Dlaczego? Odpowiedzią w tym wypadku jest (a) występowanie dużej liczby *n-gramów*, które w języku (np. -&) nie istnieją w ogóle (dla zwykłego alfabetu Vigenère'a ogromna większość możliwych *n-gramów* występuje w języku regularnie – tylko w różnej częstotliwości), (b) najczęściej w języku angielskim występują litery, więc zdecydowanie wagą dominują przesunięcia dekodujące, które najczęściej występujące znaki przenoszą na litery – a takich przesunięć dla alfabetu (3) czy (2) ((2) łączył w sobie kodowanie spacji i omawianą teraz cechę, stąd te świetne wyniki) nie ma za wiele. Powoduje to, że algorytm deszyfrujący w zasadzie operuje koniec końców na bardzo ograniczonym zbiorze – mniejszym niż dla zwykłego szyfru Vigenère'a – bowiem mnóstwo możliwości może zostać odrzuconych od razu z racji niskiej wagi. Stąd należy pamiętać, przy projektowaniu/stosowaniu szyfru, że występowanie w przestrzeni związanej z szyfrem struktur, które pozwalają bardzo łatwo zredukować przestrzeń poszukiwanych rozwiązań, znacząco osłabia moc szyfru. Szyfr taki jak ten nieźle się nada natomiast do szyfrowania rzeczy 'dziwnych'. Jeśli rzecz będzie spoza naturalnego języka, atak częstościowy, a więc i *n-gramowy* zawiedzie – choć jak już wspomnieliśmy, ta uwaga dotyczy wszystkich szyfrów. Wyniki, które uzyskałem na dłuższych alfabetach są ważne, bo pokazują, że heurystycznie wyprowadzone założenie o *liniowości* MCMC

względem s przynajmniej w dużej mierze jest poprawne, jeśli tylko poziom skomplikowania szyfru nie zwiększa się.

Już prawie koniec testów, zobaczmy jeszcze tylko jak radzą sobie metody MCMC z zakodowanym szyfrem Vigenère'a (wracamy do alfabetu (1)) tekstem, z którego wcześniej wszystkie znaki nieszyfrowane zostały usunięte – problem, który poruszyliśmy na początku podrozdziału:

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.02s	1.0	2.80s	1.0	3.70s	1.0
80	0.01s	0.75	2.90s	0.97	3.79s	0.97
120	0.01s	0.6	2.97s	0.83	3.82s	0.84
160	0.01s	0.5	3.02s	0.6	3.90s	0.64
200	0.01s	0.42	3.08s	0.5	3.99s	0.57
240	0.01s	0.34	3.15s	0.42	4.07s	0.47
280	0.01s	0.28	3.53s	0.33	4.08s	0.31
320	0.01s	0.29	3.26s	0.3	4.16s	0.29
360	0.02s	0.24	3.33s	0.2	4.17s	0.21
400	0.02s	0.21	3.38s	0.18	4.25s	0.16

Tabela 28: *Rok 1984*, fragment dł. 799, usunięte przerwy między słowami, szyfr Vigenère'a, klasyczny alfabet (1)

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.02s	0.95	2.80s	1.0	3.73s	1.0
80	0.01s	0.82	2.88s	0.96	3.76s	1.0
120	0.01s	0.69	2.94s	0.89	3.79s	0.95
160	0.01s	0.49	3.00s	0.68	3.88s	0.76
200	0.01s	0.44	3.06s	0.59	3.99s	0.61
240	0.01s	0.33	3.13s	0.42	4.23s	0.47
280	0.01s	0.32	3.85s	0.35	4.29s	0.35
320	0.01s	0.29	3.67s	0.29	4.40s	0.30
360	0.02s	0.25	3.81s	0.26	4.28s	0.25
400	0.02s	0.21	3.43s	0.19	4.28s	0.17

Tabela 29: *Ice hockey*, fragment dł. 802, usunięte przerwy między słowami, szyfr Vigenère'a, klasyczny alfabet (1)

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.01s	0.9	2.69s	1.0	3.56s	1.0
80	0.01s	0.72	2.77s	0.96	3.69s	0.98
120	0.01s	0.58	2.89s	0.69	3.75s	0.74
160	0.01s	0.48	2.88s	0.58	3.77s	0.65
200	0.01s	0.37	2.98s	0.39	4.22s	0.41
240	0.03s	0.31	3.42s	0.35	4.50s	0.32
280	0.01s	0.30	3.13s	0.32	3.94s	0.34
320	0.01s	0.27	3.14s	0.21	3.99s	0.24
360	0.01s	0.23	3.20s	0.20	4.05s	0.19
400	0.02s	0.21	3.27s	0.17	4.11s	0.20

Tabela 30: *Madame Bovary*, fragment dł. 773, usunięte przerwy między słowami, szyfr Vigenère'a, klasyczny alfabet (1)

Zobaczmy jeszcze jak to wygląda dla szyfru afinicznego:

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.20s	0.77	33.0s	1.0	43.7s	1.0
80	0.21s	0.4	34.3s	0.55	45.0s	0.63
120	0.21s	0.22	35.3s	0.20	45.8s	0.21
160	0.22s	0.08	35.9s	0.08	46.8s	0.08
200	0.23s	0.09	37.5s	0.07	58.1s	0.05
240	0.31s	0.06	49.0s	0.05	59.9s	0.02
280	0.26s	0.06	41.7s	0.02	50.4s	0.02
320	0.26s	0.02	40.5s	0.01	54.0s	0.01
360	0.27s	0.01	41.8s	0.01	53.4s	0.01
400	0.27s	0.01	41.7s	0.01	54.6s	0.01

Tabela 31: *Madame Bovary*, fragment dł. 779, usunięte przerwy między słowami, szyfr afiniczny-/rozszerzony, klasyczny alfabet (1)

Choć oczywiście moc metody jest mniejsza, to cały czas pozwala w istotny sposób poprawić uzyskane rozwiązania, co w pewnych przypadkach może być kluczowe dla skutecznego odszyfrowania.

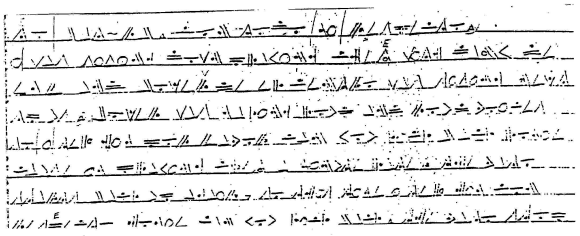
Koniec końców, wszystkie wyniki pokazują, że zastosowanie metod MCMC (wraz z teorią *n-gramów*) jest dobrym wyborem przy dekodowaniu szyfrów polialfabetycznych z opisanych rodzin – w wielu punktach to podejście może zostać jeszcze rozwinięte. Może też być jeszcze szybsze. Wybrałem dość bezpieczną stałą 10 na liczbę kroków ($10 \cdot n_k \cdot s$). Jednak przykład dekodowania *Madame Bovary* ze stałą 5 wyniki były niewiele gorsze:

DŁUGOŚĆ KLUCZA	MONOGRAM CZAS	MONOGRAM SKUTECZNOŚĆ	BIGRAM CZAS	BIGRAM SKUTECZNOŚĆ	TRIGRAM CZAS	TRIGRAM SKUTECZNOŚĆ
40	0.01s	0.92	1.34s	1.0	1.78s	1.0
80	0.01s	0.73	1.39s	0.98	1.87s	1.0
120	0.01s	0.5	1.44s	0.89	1.87s	0.91
160	0.01s	0.38	1.46s	0.65	1.90s	0.86
200	0.01s	0.38	1.49s	0.53	1.93s	0.67
240	0.01s	0.26	1.53s	0.4	1.95s	0.50
280	0.01s	0.27	1.56s	0.38	1.98s	0.52
320	0.01s	0.24	1.59s	0.28	2.03s	0.35
360	0.01s	0.22	1.62s	0.30	2.34s	0.37
400	0.02s	0.18	1.66s	0.23	2.11s	0.26

Tabela 32: *Madame Bovary* dł. 1000, klasyczny szyfr Vigenère'a, alfabet (1), mniejsze o połowę ograniczenie na liczbę kroków: $5n_k s$ zamiast $10n_k s$ jak do tej pory

Analiza takich kwestii jak dobrze umotywowany wybór odpowiednich stałych, techniki uczenia itd. jest w stanie jeszcze zwiększyć skuteczność tego podejścia. Poza tym pozostaje wiele otwartych kwestii: m.in. można spróbować rozwiązać problem z ograniczoną, ale nieustaloną długością klucza bardziej w duchu MCMC. Warto byłoby zrobić dokładną analizę, jak skuteczność algorytmu ma się do zależności między zbiorem uczącym i danymi wejściowymi (na wzór [2]) – w naszym przypadku wydaje się, że na fragmentach *Ice hockey* osiągane są często gorsze wyniki – słowniki uczyliśmy na tekstach literackich, a to jedyny 'nieliteracki' tekst. Można ponadto prawdopodobnie zwiększyć odporność algorytmu na usuwanie przerw między słowami przez nieco zmodyfikowane uczenie słowników (np. liczyć z pewną mniejszą wagą *oszukane n-gramy* w zbiorze uczącym). Na uwagę zasługuje też zagadnienie zbieżności łańcucha dla dużych długości klucza. Także, można sprawdzić działanie podejścia dla innych podobnych szyfrów np. [18].

Poza zaprezentowanymi rozwiązaniami próbowałem również problemu dekodowania, który miał postać złożenia permutacji i szyfru Vigenère'a, co można rozumieć jako szyfr Vigenère'a, ale gdy znamy tylko zbiór znaków alfabetu bez znajomości ich uporządkowania. Niestety nie udało mi się osiągnąć znaczących sukcesów. Rozwiązanie dla szyfru podstawieniowego zostało zaprezentowane w [2] [1] [3]. M.in. w pracy Diaconisa została opisana sytuacja, gdzie metody MCMC (wędrówka po permutacjach) zostały użyte do złamania szyfru podstawieniowego – którym w listach posługiwali się więźniowie – mającego postać jak na obrazku poniżej.



Rysunek 2: Szyfr więzienny

Do złamania samego szyfru permutacyjnego z powodzeniem zastosowałem opisane wcześniej w wymienionych pracach metody, jednak do złamania złożenia szyfrów konieczne było rozwinięcie tych metod. Testowałem kilka pomysłów – najbardziej obiecującym wydawał mi się oparty na wagach monogramowych – tj. wędrujemy po permutacjach, funkcją permutacji (proporcjonalną do rozkładu stacjonarnego) jest maksymalna monogramowa waga, którą można na tej permutacji alfabetu osiągnąć - jeśli długość klucza Vigenère'a jest mała, a tak w tym wypadku zakładamy, to metoda monogramowa daje dobre rezultaty. Niestety podejście to zawiodło – przynajmniej przy zadanych ograniczeniach czasowych (nie można było wykonać aż tak wielu kroków jak w dekodowaniu szyfru Vigenère'a czy podstawieniowego, bo każdy krok tutaj miał większą złożoność czasową – $O(s \cdot m)$, jak już wcześniej policzyliśmy). Drugim podejściem była próba przyspieszenia zbieżności przez odpowiedni stan startowy i specjalnie dobrane kroki. Zauważyć można, że jeśli zaczniemy od stanu, który jest zgodny z częstościami występowania pojedynczych liter, powinniśmy robić raczej krótkie transpozycje (tj. np. 'E' nie powinna być w tekście najrzadziej występującą literą, raczej powinna być dość blisko pierwszego miejsca). Mimo że na problemie samych permutacji zastosowanie tylko 'krótkich' transpozycji dawało nieco szybszą zbieżność (choć mniej dokładną), to nie udało się ulepszyć w ten sposób algorytmu Vigenère+permutacja (ideą było tak samo użycie wagi monogramowej jako funkcji stanu, ale teoretycznie w bardziej uporządkowanym spacerze losowym). Możliwe, że jest możliwość wypracowania lepszego podejścia MCMC do tego problemu, niestety w obrębie tej pracy nie udało się to – być może zajmę się tym jeszcze w przyszłości. Testowałem poza tym dla tego problemu alfabety jedynie bez spacji. Jak już wiemy, kodowanie spacji, dla tekstów w języku angielskim, przy omawianych tu typach szyfrów daje atakującemu szyfr duży handicap. Tak więc warto będzie prawdopodobnie do tego tematu wrócić.
Mimo wszystko udało się skutecznie zastosować MCMC do szyfrów z rodziny Vigenère'a i pochodnych i przyspieszyć dzięki temu podejściu rozwiązanie w porównaniu z algorytmami deterministycznymi, co było celem tej sekcji. Według mojej wiedzy żadne z opisanych tu podejść do szyfru Vigenère'a i pochodnych nie było stosowane. Ponadto problem łamania szyfru Vigenère'a z długim kluczem wydaje się mało zbadany. Natknąłem się tylko na pracę [22] – na swoich zbiorach danych osiągnąłem lepsze wyniki mniej skomplikowanym algorytmem, jednak rzetelne porównanie nie jest możliwe – testy w [22] przeprowadzane były na fragmentach tekstu, do którego nie mam dostępu.

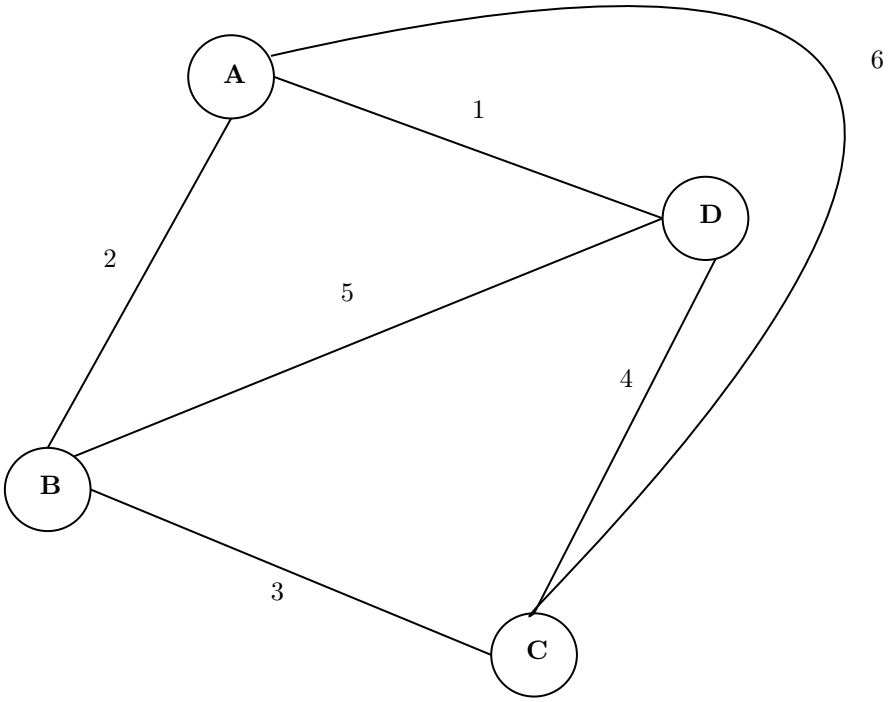
Podsumowując, należy zaznaczyć, że podejście MCMC do dekodowania zostało z sukcesem zastosowane na nowej rodzinie szyfrów.

5 Problem komiwojażera

W rozdziale tym omówię podejście MCMC do rozwiązania problemu komiwojażera. Następnie zaprezentuję jego wyniki na benchmarkowym zbiorze instancji problemu.

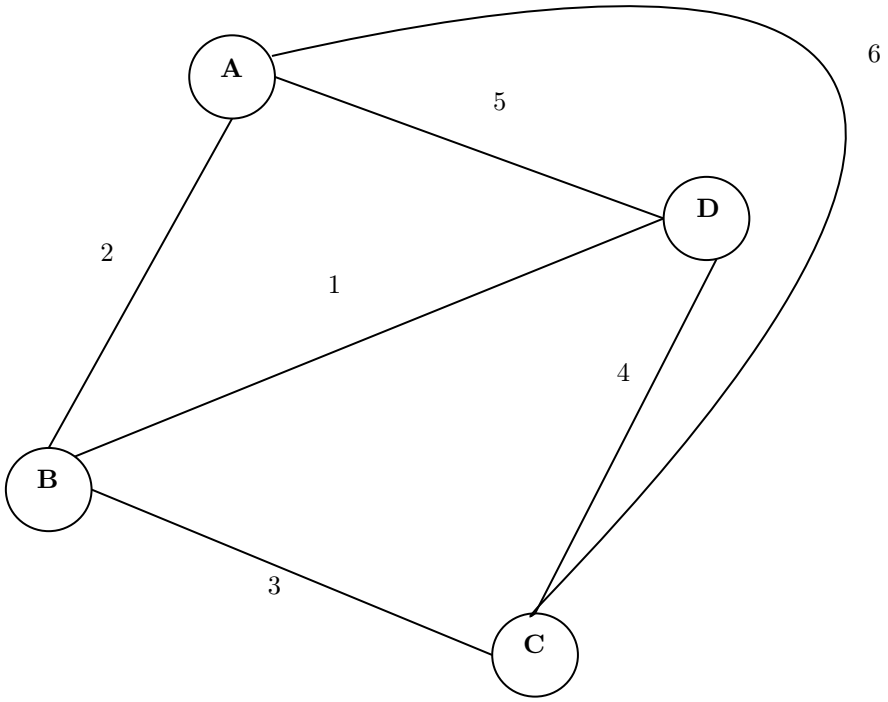
5.1 Wprowadzenie, opis problemu

Wróćmy do grafu – ważonego – i zobaczmy jego przykład jak poniżej:



Rysunek 3: Jaką trasę powinien tu wybrać komiwojażer?

Liczby nad krawędziami nazywamy ich wagami i możemy o nich myśleć jako o odległościach między miastami (wierzchołkami) między którymi biegną krawędzie (A,B,C,D). Wyobraźmy sobie teraz, że jesteśmy tytułowym komiwojażerem, chcemy przedstawić naszą ofertę we wszystkich miastach w jak najkrótszym czasie (czas to pieniądz!), startując z naszego rodzinnego miasta A, przechodząc przez pozostałe miasta tylko raz i na koniec wracając do A. Innymi słowy, chcemy zminimalizować sumę odległości na drodze przez wszystkie miasta (przez każde miasto przechodząc dokładnie raz), na końcu wracając do miasta startowego. Taka 'podróż' nazywana jest cyklem Hamiltona. Chcemy więc znaleźć cykl Hamiltona o najmniejszej długości, oczywiście wybór miasta początkowego nie ma znaczenia – jak to w cyklu. W powyższym wypadku odpowiedź jest łatwa do sprawdzenia – zbiór możliwych cykli jest mały ($\frac{4!}{4} = 3! = 6$). Ponadto rozwiązanie 'widać' – cykl ABCD zawiera 4 krawędzie o najmniejszych wagach, więc w oczywisty sposób jest rozwiązaniem problemu. Jednak już dla grafu poniżej z zamienionymi wagami krawędzi rozwiązanie nie rzuca się już aż tak w oczy:



Rysunek 4: A jaką tu?

Tym razem będzie to DBAC. Zanim przejdziemy do dalszych rozważań sformalizujemy pojęcia grafu (choć intuicyjnie jest ono jasne) i cyklu Hamiltona. Zakładamy, że grafy, na których pracujemy, są pełne – jeśli nie byłoby krawędzi między pewnymi różnymi wierzchołkami to przyjmujemy po prostu, że jest krawędź o wadze ∞.

Definicja 5.1.1.
Grafem ważonym nazwiemy zbiór wierzchołków V, $|V| = n$ i krawędzi $E \subseteq V \times V$, tak że każdej krawędzi e jest przypisana długość (waga) $w_e \in \mathbb{R} \cup \{+\infty\}$.

W obecnych rozważaniach przyjmujemy, że nie ma krawędzi między wierzchołkiem a tym samym wierzchołkiem (albo jej waga to zero), ponadto graf jest symetryczny, więc krawędź ij utożsamiamy z ji, podobnie ich wagi. Zadaniem w problemie komiwojażera jest odnalezienie permutacji σ wierzchołków (dalej przyjmujemy ich numerację $1, ...n$), takiej że $\sum_{i=1}^{n} w_{\sigma(i)\sigma(i+1)}$ ma wartość minimalną przy czym przyjmujemy $n + 1 := 1$. Szukamy więc dla danego grafu:

$$\sigma_{min} = \arg\min_{\sigma \in S_n} \sum_{i=1}^{n} w_{\sigma(i)\sigma(i+1)}$$

Teraz, problem ten jest *trudny*. Konkretnie NP-trudny, to znaczy, że każdy problem z klasy NP (problemów, dla których rozwiązanie jest weryfikowalne w wielomianowym czasie) daje się do niego sprowadzić również wielomianową transformacją. Z tego powodu, nawet gdyby, co mało prawdopodobne, okazało się, że $P = NP$ (czyli że wszystkie problemy z wielomianową weryfikacją mają też wielomianowe rozwiązanie), nie daje to nam pewności, że problem komiwojażera da się rozwiązać w wielomianowym czasie. Jeśliby się natomiast dało, wtedy byłoby $P = NP$, co jest warte milion dolarów. Tak więc, nie są znane deterministyczne algorytmy o wielomianowej złożoności czasowej rozwiązujące postawiony problem – w przeciwieństwie do problemu dekodowania szyfru Vigenère'a

w postawionej przeze mnie wersji, gdzie takie rozwiązania istniały, jednak były *tylko* nieoptymalne czasowo. Użyjemy więc znowu metod Markov Chain Monte Carlo. Tym razem jednak, jak już wcześniej wspomnieliśmy, poza klasycznym rozwiązaniem z użyciem JŁM sprawdzimy też jedno z klasycznych rozwiązań prezentujących podejście zbieżnościowe – by 'zwiększyć' zbieżność – i osiągnąć zatrzymanie języka w jakimś do pewnego stopnia optymalnym stanie – musimy zrezygnować z jednorodności. Podejście to nosi nazwę symulowanego wyżarzania.

5.2 Niejednorodne łańcuchy Markowa. Symulowane wyżarzanie

Pomysł, który jest zastosowany do osiągnięcia zbieżności, ma swoje źródła w fizyce statystycznej. Możemy myśleć o naszych permutacjach (czyli potencjalnych rozwiązaniach) jako o pewnych stanach energetycznych, które są do obsadzenia przez chaotycznie poruszającą się cząstkę (cząstki). Cząstka zajmuje dany stan z prawdopodobieństwem związanym z poziomem energetycznym tego stanu – im wyższa energia stanu, tym mniejsze prawdopodobieństwo, konkretnie: stosunek prawdopodobieństw obsadzenia dwóch stanów energetycznych σ_i σ_j o energiach E_i oraz E_j odpowiednio wynosi:

$$\frac{e_i}{e_j} = e_{ij} = \exp\left(\frac{E_j - E_i}{kT}\right)$$

stąd ogólnie prawdopodobieństwo obsadzenia danego stanu energetycznego i przy temperaturze T wynosi:

$$p_i = \frac{\exp(\frac{-E_i}{kT})}{c}$$

gdzie $k > 0$ jest pewną stałą (stała Boltzmanna) a $T \geq 0$ – temperaturą układu, natomiast c stałą normalizującą, tak żeby uzyskać rozkład prawdopodobieństwa. Widać pewne zależności: generalnie im energia niższa tym prawdopodobieństwo wyższe, przy T dążącym do nieskończoności każdy stan staje się jednakowo prawdopodobny, natomiast przy $T = 0$ całe prawdopodobieństwo skupia się w stanie o minimalnej energii (skoro temperatura jest zerowa, to układ, a więc i cząstka ma minimalną energię). Ma to oczywiście swoje odzwierciedlenie w rzeczywistości – rzeczywiście im temperatura wyższa, tym cząstki są bardziej ruchliwe, tym więcej się zderzają ze sobą, tym większy jest 'przepływ' energii, więc tym więcej stanów energetycznych zajmują. W zerze absolutnym cząstki natomiast zastygają w bezruchu – posiadają najniższą możliwą energię.
Tak jak więc zostało wspomniane, możemy myśleć o naszych permutacjach jako o stanach energetycznych do obsadzenia przez cząstkę. Im wyższa energia, czyli większa suma odległości w cyklu Hamiltona odpowiadającemu permutacji, tym prawdopodobieństwo niższe. I odwrotnie, im mniejsza suma odległości, tym prawdopodobieństwo wyższe. Wiążemy więc łańcuch, który konstruujemy, z rozkładem Boltzmanna, mianowicie chcemy, aby przy ustalonej temperaturze łańcuch miał ten właśnie rozkład stacjonarny.

Przy ustalonej temperaturze, p_i odpowiada więc π_i z algorytmu Metropolisa. Do konstrukcji przejścia z i do j używamy więc stosunków $\frac{p_j}{p_i}$ tak jak ogólnie używa się stosunków $\frac{\pi_j}{\pi_i}$, czyli:

$$p_{ij} = \frac{p_j}{p_i} = e_{ji} = \exp\left(\frac{E_i - E_j}{kT}\right)$$

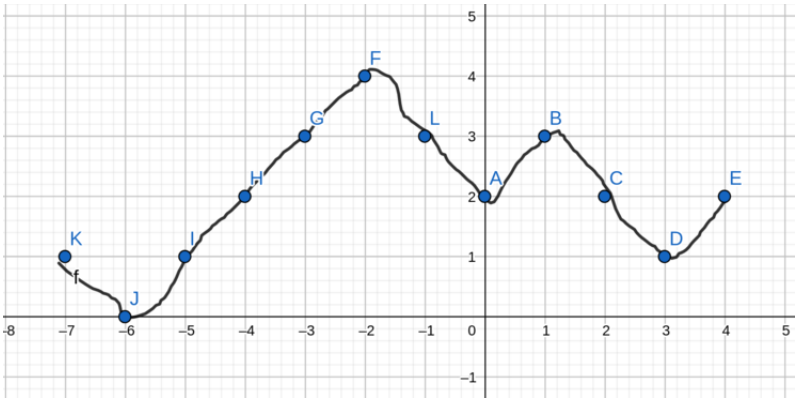
Chcemy zdefiniować problem tak, aby przy temperaturze dążącej do zera, poszukiwane przez nas rozwiązanie minimalizujące odległość (czyli maksymalizujące prawdopodobieństwo w rozkładzie Boltzmanna), było właśnie stanem, który zajmie cząstka. Tak więc wiążemy sumę odległości w cyklu Hamiltona z energią (tj. suma odległości w danej permutacji i, jest energią tej permutacji: E_i) jak wyżej i dostajemy taki łańcuch na bazie algorytmu Metropolisa-Hastingsa (prosta modyfikacja kroku pozwala nam wygenerować niejednorodny łańcuch):

Algorytm 6: Symulowane wyżarzanie

Zacznij w dowolnie wybranym stanie (może być ustalony lub np. wylosowany jednostajnie) $X_0 := \sigma_0 \in S_n$
dla: $n = 0, 1, 2, 3...$
  wylosuj Z – kandydata na nowy stan zgodnie z pewnym rozkładem $q_{X_n \cdot}$ (nieredukowalna macierz generująca kandydatów, taka że powstały na jej podstawie łańcuch jest nieredukowalny i nieokresowy);
  $p_{X_n Z} = \exp(\frac{E_{X_n} - E_Z}{t_n})$;
  wylosuj $V \sim U[0, 1]$;
  jeśli: $V \leq \min(1, p_{X_n Z})$
    $X_{n+1} := Z$;
  w przeciwnym razie:
    $X_{n+1} := X_n$;

t_n – jest parametrem wyżarzania bądź wychładzania (zadającym jego *harmonogram* tzw. *cooling schedule*) – temperaturą w czasie n, tj. mówi, jak zmieniamy (zmniejszamy) temperaturę w czasie. Zwróćmy uwagę, że gdybyśmy przyjęli $t_n \equiv const$, dostajemy wtedy zwykły JŁM o rozkładzie stacjonarnym odpowiadającym rozkładowi Boltzmanna. W t_n ukryta jest oczywiście też stała k. W praktyce oczywiście znowu na obie strony nierówności w *if*-ie nakładamy logarytm, co jest rzecz jasna o wiele praktyczniejsze obliczeniowo. W idei tej metody zawarte jest założenie $t_n \downarrow 0$ – cały proces przypomina *wyżarzanie* (schładzanie) metalu, stąd nazwa. Można zastanawiać się, kiedy w istocie zajdzie zbieżność do globalnego minimum względem sumy odległości cyklu Hamiltona odpowiadającego permutacji. Łatwo udowodnić, że przynajmniej do minimum lokalnego zbieżność zachodzi prawie na pewno. Jak uzyskiwać zbieżność do minimum globalnego do pewnego stopnia pokazał Hajek [33] – warto rozważać t_n postaci $t_n = \frac{c}{\log(n+2)}$ (u Hajka jest to $\frac{c}{\log(n+1)}$, jednak my numerujemy łańcuch od zera) dla pewnej stałej $c > 0$, która powiązana jest z pewnymi szczególnymi własnościami rozważanego grafu ('głębokościami' minimów lokalnych). Dla stałych większych lub równych od pewnego d^* (i tylko dla takich) zachodzi zbieżność prawie na pewno do globalnego minimum (jednego z nich, jeśli jest więcej). W praktyce (1) dokładne wyznaczenie d^* może być bardzo trudne – w zasadzie tak samo jak sam problem komiwojażera, (2) d^* może być tak duże, że zbieżność jest bardzo powolna (przez długi czas łańcuch zachowuje się jakby stany były losowane w sposób niemal jednostajny), przez co w realnym czasie nic nie zyskujemy. Powiedzmy tylko obrazowo o głębokości minimum i czym jest d^*.

Spójrzmy na poniższy rysunek:



Rysunek 5: Głębokość minimów

Na osi Y mamy wartości energii. Lokalnymi minimami są (rysunek jest niedoskonały, więc w rzeczywistości minimami są punkty zbliżone do nich, ale utożsamiamy minima z tymi punktami, podobnie jak wartości ich utożsamiamy z najbliższymi liczbami całkowitymi) A, D, J (formalnie $x = -6, 0, 3$). J jest również minimum globalnym. O głębokości minimum lokalnego myślimy jako o najmniejszej różnicy poziomów (energii) – gdzie poruszamy się w sposób ciągły po wykresie – którą należy pokonać (czyli najmniejszej ilości energii, którą należy dodatkowo zyskać), aby móc trafić do

mniejszego minimum lokalnego. Tak więc energia minimum A to ~ 2. Najmniej energii potrzeba, by dojść do D, wystarczy przejść przez B o energii 3, a więc zyskać jednostkę energii (następnie już schodzimy na niższe poziomy energetyczne, więc nie jest potrzebna dodatkowa energia). Tak więc głębokość A wynosi 1. Głębokość D natomiast wynosi 3, by dojść do J – minimum o mniejszej energii trzeba przejść przez F o energii 4. J natomiast jest minimum globalnym, przyjmujemy więc głębokość równą ∞. Dobrą analogią jest tu głębokość przełęczy, jak zresztą widać na rysunku.

Podobnie sprawa się ma z minimami lokalnymi w łańcuchu Markowa. Głębokość minimum lokalnego (czyli takiego stanu, dla którego wszyscy *kandydaci* na następny stan zgodnie z macierzą generującą mają większą energię), to minimalna energia, którą trzeba zyskać, na drodze do mniejszego minimum, gdzie wybieramy spośród dróg, które mają niezerowe prawdopodobieństwa wg macierzy przejścia tego łańcucha. Przechodzenie tylko do sąsiadów (czyli stanów z niezerowym prawdopodobieństwem przejścia do nich) jest odpowiednikiem poruszania się w sposób ciągły z przykładu powyżej.

Nasza d^* to maksymalna spośród głębokości wszystkich minimów lokalnych (niebędących globalnymi). W praktyce, tak jak już zostało wspomniane d^* może być na tyle duże, że nie osiągamy z powodu jego użycia zbieżności w żadnym rozsądnym czasie (np. dla $c = d^* = 100$ aż do momentu e^{100} łańcuch jest 'bardziej ruchliwy' niż zwykły JŁM z $t_n \equiv 1$, bowiem temperatura startowa jest bardzo wysoka, a jej spadek stosunkowo powolny). Z pracy Hajka wynika jednak też, że przyjmując dane c wykluczamy jako potencjalne punkty zbieżności wszystkie minima lokalne o głębokości $< c$ (ogólnie im mniejsza głębokość, tym mniejsze prawdopodobieństwo skończenia w danym minimum, bowiem tym łatwiej jest się z niego wtedy 'odkopać', tym mniej energii dodatkowo musimy wtedy 'pozyskać'), stąd wiemy, że raczej nie trafimy przynajmniej na najpłytsze z minimów – a takie minima są – jak widać na przykładzie A z rysunku – najmniej korzystne.

d^* w przypadku problemu komiwojażera można łatwo oszacować z góry przez:

$$n \cdot (\max_{ij} w_{ij} - \min_{ij} w_{ij}),$$

największa suma odległości jest bowiem nie większa od:

$$n \cdot \max_{ij} w_{ij},$$

a najmniejsza nie mniejsza niż

$$n \cdot \min_{ij} w_{ij}.$$

Jednak zwykle jest to oszacowanie bardzo na wyrost, ogólnie d^* może być zwyczajnie bardzo duże - jeśliby je przyjąć, nie osiągniemy zbieżności w rozsądnym czasie. Ostatecznie dla testów przyjmujemy $c = 3$, co jest kompromisem pomiędzy odpowiednio długim wychładzaniem a szybką zbieżnością równoważną z możliwością przeprowadzenia testów na względnie dużej liczbie instancji problemu (na standardowym komputerze) i mimo wszystko powinno nas doprowadzić do suboptymalnych rozwiązań na zadowalającym poziomie.

5.3 Macierze generujące. Omówienie kroków algorytmu

Algorytm został już przedstawiony w poprzednim podrozdziale, jedynymi szczegółami do ustalenia pozostają (1) wybór stanu startowego, (2) wybór macierzy generującej kandydatów, (3) wybór *cooling schedule*, czyli ciągu t_n. O (1) zakładamy, że stan startowy jest losowany jednostajnie ze zbioru wszystkich permutacji. Dla (2) będziemy badać dwie możliwości, które przedstawione zostaną poniżej. Jeśli chodzi o (3), to tak jak wspomnieliśmy już wcześniej, będziemy sprawdzać zachowanie algorytmu dla $t_n = \frac{3}{\log(n+2)}$ oraz $t_n = 1$, czyli klasycznego JŁM. Przejdźmy więc do opisu dwóch opcji macierzy generujących, które nazwiemy **K1** i **K2** ("K"jak kandydat).

1. **K1**
   Wybieramy jednostajnie losowo dwa miasta (wierzchołki) z aktualnego stanu i zamieniamy je miejscami, np.:
   – aktualny stan A B C E D F;
   – losowo wybrane zostają A i E;

– kandydat to E B C A D F.

W praktyce można wybierać po prostu liczby od 1 do n (czy od 0 do $n-1$, gdy numerujemy od 0) i zamieniać wierzchołki odpowiadające tym liczbom w aktualnej permutacji. Zwróćmy uwagę, że po takiej zmianie w naszym cyklu Hamiltona zmieniły się co najwyżej 4 krawędzie – te do których styczne były zamienione wierzchołki. By obliczyć zaktualizowaną sumę odległości wystarczy więc odjąć 4 'stare' długości krawędzi i dodać 4 'nowe'. W naszym przypadku odejmujemy:
FA, AB, CE, ED
natomiast dodajemy:
FE, EB, CA, AD.
Mamy więc co najwyżej 4 dodawania i odejmowania, zamiana miejscami elementów zaś to koszt stały, więc dostajemy stały $O(1)$ koszt kroku. Krok jest więc *szybki*, co jest kwestią bardzo istotną dla algorytmu.

2. **K2** Bazą dla opcji numer 2 również jest *swap* (zamiana dwóch elementów), z tym, że tym razem odwrócimy też ciąg wierzchołków pomiędzy nimi. Spójrzmy na przykład:
– aktualny stan A B C E D F;
– losowo wybrane zostają A i E w przeciwieństwie do opcji nr 1 kolejność ma znaczenie);
– kandydat to E C B A D F (uwaga: gdyby wybrane zostały E i A (czy też odpowiadające im pozycje), to kandydatem byłby E B C A F D).

   W **K1** mieliśmy $\frac{n(n-1)}{2}$ kandydatów, tu jest ich dwa razy więcej: $n(n-1)$, gdyż jak to przedstawiliśmy, kolejność ma znaczenie. Zwróćmy teraz uwagę, że koszt obliczenia nowej sumy odległości (czy też energii wg nomenklatury wyżarzania) jest mały! Tym razem zmieniają się tylko wagi co najwyżej dwóch krawędzi – odwrócenie ciągu nic nie zmienia w sumie wag z racji zakładanej symetrii grafu. W naszym przykładzie do zaktualizowania sumy odległości wystarczy odjąć:
   – AF i ED;
   oraz dodać:
   – EF i AD.
   Koszt aktualizacji *funkcji stanu* (tak również możemy nazywać energię/sumę odległości) jest więc mały i stały. Zostaje kwestia odwrócenia ciągu pomiędzy. Tu niestety, jeśli oznaczymy sobie przez x jego długość (równą mniej więcej różnicy pozycji, które *swapujemy*), potrzebujemy $\sim x/2$ operacji. Średnio pozycje są oddalone o $n/2$, więc jest to $n/4$. Zwróćmy uwagę jednak, że odwracanie ciągu jest konieczne jedynie wtedy, gdy kandydat został przyjęty (nie potrzebujemy go do obliczenia *update'u* funkcji stanu). Na ogół więc również krok w wersji nr 2 jest szybki, zwykle również n nie jest aż tak duże (w realnych zastosowaniach $< 10^5$. Intuicyjnie zaś zwiększa zasięg łańcucha, jako że w pewnym sensie w jednym kroku zmieniamy więcej (mimo to zachowując 'ciągłość' kroku, co również jest konieczne). Jak się okaże później, właśnie mały zasięg **K1** nie pozwoli jej używać w bardzo efektywny sposób – stąd to **K2** jest w tym wypadku opcją preferowaną. **K2** ma swoją nazwę - jest to krok *opt-2*, zaproponowany jeszcze w latach 50. w [15] i [16].

Na koniec rozważań w temacie macierzy przejść wypada zauważyć, że na bazie tego samego argumentu, którego użyliśmy przy okazji dekodowania, powstały łańcuch jest nieokresowy (jeśli tylko rozkład stacjonarny nie jest jednostajny, co jest oczywistym założeniem, zauważmy, że pojęcia jak nieredukowalność i nieokresowość w naturalny sposób rozszerzają się na niejednorodne łańcuchy Markowa), jest również nieredukowalny, każdą permutację da się bowiem przedstawić jako złożenie transpozycji liczb sąsiednich. Mamy więc w tym momencie już wszystkie kroki algorytmu, pozostaje testowanie różnych opcji, do czego przejdziemy w następnym rozdziale.

5.4 Kilka słów o zbiorze testowym

Zaprezentowane algorytmy będziemy testować na zbiorze testowym TSPLIB [4]. Historia jego sięga 1991 roku [37], kiedy to Gerhard Reinelt z uniwersytetu w Heidelbergu opublikował zbiór danych wejściowych mających służyć jako benchmark, dzięki któremu różne grupy pracujące nad

metodami optymalizacji, mogły porównywać osiągane przez siebie wyniki. Biblioteka była z czasem uzupełniana kolejnymi instancjami problemu o różnej wielkości wejścia (liczbie wierzchołków grafu). Instancje problemu pochodziły ze świata rzeczywistego - problem komiwojażera występuje w nim oczywiście w różnych wersjach bardzo naturalnie.
Na początku dla większości instancji nie były znane rozwiązania optymalne, jednak z czasem, cytując za profesorem Reineltem (2007):
When I published TSPLIB more than 10 years ago, I expected that at least solving the large problem instances to proven optimality would pose a challenge for the years to come. However, due to enormous algorithmic progress all problems are now solved to optimality!!
Dzięki temu dla wszystkich instancji symetrycznego problemu komiwojażera z TSPLIB mamy rozwiązania *prawdziwie* optymalne, dzięki czemu możemy do nich porównywać osiągane przez własne implementacje algorytmów wyniki.

Do zbioru TSPLIB należy największa dotychczas rozwiązana nietrywialna instancja symetrycznego problemu komiwojażera licząca 85900 wierzchołków [38]. Do rozwiązania wielu z instancji użyte zostały superkomputery w połączeniu z różnymi pomysłowymi heurystykami.

Nazwy instancji problemów w tym zbiorze są tworzone w ten sposób, że ich pierwsza część oddaje rzeczywiste pochodzenie (np. berlin52 – problem dotyczy komunikacji w Berlinie, *att532*, sieć przesyłowa AT&T), druga część zaś wskazuje jak wiele wierzchołków jest w danej instancji. Zajmę się w symulacjach instancjami: *dsj1000, att532, kroA150, berlin52*.

5.5 Symulacje

Liczbę kroków pokonanych przez nasz łańcuch ograniczamy przez $10n^2$, jeśli $10 \cdot n^2 < 10^7$, gdzie n to liczba miast w instancji problemu. W przeciwnym wypadku za ograniczenie przyjmujemy 10^7 (nie dotyczy to instancji problemów, które będziemy omawiać tu, jednak przeprowadzałem też takie testy). To właśnie tę liczby przyjmuje za ograniczenie liczby kroków – każdy *swap* będzie miał wtedy szansę wykonać się 10 razy (a nawet trochę więcej). Sprawdzane będą dwie opcje macierzy przejścia opisane w poprzednim rozdziale: **K1** i **K2**, $t_n = 1$ – czyli jednorodny łańcuch Markowa oraz $t_n = \frac{3}{\log(n+2)}$. Dla $t_n \equiv 1$ wędrówkę kontynuujemy aż do górnego ograniczenia liczby kroków, dla drugiego t_n przyjmujemy *kryterium zbieżności*. Mianowicie, jeśli łańcuch pozostał w tym samym stanie przez $0,9 \cdot n \log_2 n$, gdzie n jest liczbą miast/wierzchołków w instancji problemu, kroków, uznajemy, że zbiegł. $n \log_2 n$ pojawia się w tym ograniczeniu z tego powodu, że jest to z dokładnością do stałej czas, w którym teoretycznie można dojść do dowolnej konfiguracji (dojście do permutacji jest w zasadzie równoważne z problemem sortowania, stąd $n \log_2 n$ pojawia się dość naturalnie, 0.9 przyjąłem sobie odgórnie).

Za rozwiązanie dostarczone przez algorytm uznajemy, jak poprzednio, *najlepszy* odwiedzony stan. Dla każdej z opcji podamy, w którym kroku został osiągnięty, jaką dostaliśmy dla niego długość trasy (*rozwiązanie*) i jaki jest stosunek tej długości do optymalnej, a także jak długi był przebieg algorytmu. Dla $t_n \equiv \frac{3}{\log(n+2)}$ podamy również do stanu o jakiej wartości funkcji (sumy odległości na zadanej przezeń drodze) łańcuch *zbiegł* (jeśli zbiegł), czyli jaki był jego ostatni stan, jeśli kryterium zbieżności zostało osiągnięte

Testy przy ustalonej instancji, konfiguracji t_n i macierzy przejścia przeprowadzamy dwa razy, tak by łańcuch miał okazję wystartować z dwóch różnych punktów. Dla porównania i pokazania zysku, jaki daje MCMC, przedstawię też najlepszy stan osiągnięty w ciągu 10000 niezależnych losowań z permutacji wierzchołków dla rozkładu jednostajnego. Wyniki testów przedstawię dla czterech wybranych instancji problemu z TSPLIB [4] z różnymi liczbami wierzchołków. Instancje problemów są zatytułowane tak, jak opisałem to w poprzednim podrozdziale: przetestujemy instancje o: 1000, 532, 150, 52 wierzchołkach. Dla każdej z instancji na początku podajemy rozwiązanie optymalne wg TSPLIB.

5.5.1 dsj1000

W tej instancji mamy 1000 wierzchołków i rozwiązanie optymalne zadające cykl długości 18660188.0. Zobaczmy, jak poradziły sobie opisane algorytmy.

NR	t_n	ROZW. UZYSKANE W KROKU	ROZW.	ROZW./ OPTYMALNE	L. KROKÓW	ZBIEŻNOŚĆ DO	CZAS
1	$= 1$	9992464	83143199.26	4.45	10000000	-	66.9s
1	$= \frac{3}{\log(n+2)}$	1909253	96085056.78	5.14	1918223	96085056.78	14.1s
2	$= 1$	9996183	81799224.84	4.38	10000000	-	67.2s
2	$= \frac{3}{\log(n+2)}$	2140958	88704358.44	4.75	2149928	88704358.44	15.8s

Tabela 33: **K1**, dsj1000

NR	t_n	ROZW. UZYSKANE W KROKU	ROZW.	ROZW./ OPTYMALNE	L. KROKÓW	ZBIEŻNOŚĆ DO	CZAS
1	$= 1$	5640347	21233340.16	1.13	10000000	-	72.0s
1	$= \frac{3}{\log(n+2)}$	5840704	20931010.98	1.12	10000000	-	77.8s
2	$= 1$	4621762	20966014.89	1.12	10000000	-	68.2s
2	$= \frac{3}{\log(n+2)}$	4494771	20838187.42	1.11	10000000	-	82.2s

Tabela 34: **K2**, dsj1000

Najlepszy wynik z 10000 jednostajnych losowań natomiast to: 527495696.62, a losowanie trwało 54.15s. Wynik uzyskany w ten sposób jest ok. **60** razy gorszy od optymalnego.

5.5.2 att532

W instancji są, jak widać, 532 wierzchołki, optymalne rozwiązanie wg danych z TSPLIB to: 27686.0.

NR	t_n	ROZW. UZYSKANE W KROKU	ROZW.	ROZW./ OPTYMALNE	L. KROKÓW	ZBIEŻNOŚĆ DO	CZAS
1	$= 1$	2757277	74449.00	2.68	2830240	-	21.9s
1	$= \frac{3}{\log(n+2)}$	544654	92308.00	3.33	548990	92308.00	4.74s
2	$= 1$	2826828	76772.00	2.77	2830240	-	21.7s
2	$= \frac{3}{\log(n+2)}$	369958	85992.00	3.10	374294	85992.00	3.08s

Tabela 35: **K1**, att532

NR	t_n	ROZW. UZYSKANE W KROKU	ROZW.	ROZW./ OPTYMALNE	L. KROKÓW	ZBIEŻNOŚĆ DO	CZAS
1	$= 1$	1504333	31667.00	1.14	2830240	-	19.0s
1	$= \frac{3}{\log(n+2)}$	1289131	30830.00	1.11	2830240	-	21.2s
2	$= 1$	2473277	31255.00	1.12	2830240	-	18.9s
2	$= \frac{3}{\log(n+2)}$	1048840	31809.00	1.14	2830240	-	21.2s

Tabela 36: **K2**, att532

Z naszych 10000 losowań zgodnych z rozkładem jednostajnym najlepszym wynikiem, który udało się uzyskać był cykl zadający długość trasy równą 469337.0, a więc **16.95** razy gorszą niż opty-

malna - w czasie 17.55s.

5.5.3 kroA150

W instancji jest 150 wierzchołków, optymalne rozwiązanie to 26524.0.

NR	t_n	ROZW. UZYSKANE W KROKU	ROZW.	ROZW./ OPTYMALNE	L. KROKÓW	ZBIEŻNOŚĆ DO	CZAS
1	$= 1$	189050	51646.54	1.94	225000	-	1.50s
1	$= \frac{3}{\log(n+2)}$	27091	63529.15	2.39	28067	63529.15	0.19s
2	$= 1$	97227	54829.62	2.06	225000	-	1.35s
2	$= \frac{3}{\log(n+2)}$	43274	61089.11	2.30	44250	61089.11	0.30s

Tabela 37: **K1**, kroA150

NR	t_n	ROZW. UZYSKANE W KROKU	ROZW.	ROZW./ OPTYMALNE	L. KROKÓW	ZBIEŻNOŚĆ DO	CZAS
1	$= 1$	103718	29423.10	1.10	225000	-	1.38s
1	$= \frac{3}{\log(n+2)}$	75045	29385.76	1.10	225000	-	1.43s
2	$= 1$	62128	29904.73	1.12	225000	-	1.53s
2	$= \frac{3}{\log(n+2)}$	60307	28546.87	1.07	225000	-	1.47s

Tabela 38: **K2**, kroA150

Metoda jednostajnego losowania dała trasę o długości 221376.09, co jest około **8** razy gorszym wynikiem niż dla trasy optymalnej. Wynik ten został uzyskany w 2.74s

5.5.4 berlin52

W instancji mamy 52 wierzchołki z optymalnym rozwiązaniem 7542.0.

NR	t_n	ROZW. UZYSKANE W KROKU	ROZW.	ROZW./ OPTYMALNE	L. KROKÓW	ZBIEŻNOŚĆ DO	CZAS
1	$= 1$	19578	9188.86	1.21	27040	-	0.14s
1	$= \frac{3}{\log(n+2)}$	3186	12714.79	1.68	3453	12714.79	0.01s
2	$= 1$	12248	10282.28	1.36	27040	-	0.14s
2	$= \frac{3}{\log(n+2)}$	2652	11732.84	1.55	2919	11732.84	0.01s

Tabela 39: **K1**, berlin52

NR	t_n	ROZW. UZYSKANE W KROKU	ROZW.	ROZW./ OPTYMALNE	L. KROKÓW	ZBIEŻNOŚĆ DO	CZAS
1	$= 1$	6790	8593.15	1.13	27040	-	0.13s
1	$= \frac{3}{\log(n+2)}$	9062	8399.97	1.11	27040	-	0.15s
2	$= 1$	6879	7960.15	1.05	27040	-	0.14s
2	$= \frac{3}{\log(n+2)}$	9440	8151.08	1.08	27040	-	0.15s

Tabela 40: **K2**, berlin52

Metodą jednostajnego niezależnego losowania dostaliśmy trasę o długości 23004.48 (ok. **3** razy gorszą od optymalnej) w czasie 0.71s.

5.5.5 Podsumowanie wyników

Testy przeprowadzałem na większej liczbie instancji problemu, jednak powyższe wyniki są reprezentatywne. Uzyskanie ~ 1.1 razy gorszych rozwiązań było wynikiem, wokół którego oscylowaliśmy w każdym przypadku dla **K2**, która to opcja okazała się najlepszą. To dobry wynik – mówiąc obrazowo: jeśli optymalna podróż komiwojażera miałaby trwać tydzień, to przyjmując naszą propozycję, wróci on do swojego rodzinnego miasta tego samego dnia, co w wersji optymalnej, tyle że po południu albo wieczorem zamiast rano.

Opcja **K1** dawała sporo gorsze wyniki. Wydaje się, że ruch uzyskany za jej pomocą zmieniał zbyt mało, przez co łańcuch mógł utykać na dłużej w 'mini' minimach lokalnych. Z drugiej strony dzięki temu **K1** w przeciwieństwie do **K2** była w stanie spełnić kryterium zbieżności, co choć zabierało rozwiązaniu optymalności, znacząco przyspieszało czas działania.

Jak widać, dla **K2** uzyskanie zbieżności dla *symulowanego wyżarzania* było wydarzeniem rzadkim przy przyjętym kryterium. Prawdopodobnie można by osłabić kryterium – zazwyczaj *najlepszy* stan był osiągany sporo przed końcem wędrówki wynikającym z przyjętych ograniczeń. Można by również przyspieszyć wyżarzanie, jest to jednak oczywiście delikatna kwestia – jeśli przyspieszymy je za bardzo, wyniki mogą być odległe od optymalnych.

Za każdym razem, zarówno opcja **K1**, jak i **K2** dawała wyniki znacznie lepsze niż metoda jednostajnego wielokrotnego losowania, stąd widać, że użycie tych metod ma znaczenie (ciągła i mimo że losowa, to w pewien sposób uporządkowana wędrówka ma większy sens niż 'skakanie' po wszystkich stanach).

Stałe dla tego rozwiązania ustaliłem po niewielkiej liczbie testów. Drobne szczegóły implementacyjne są jednak istotnym elementem wpływającym na ostateczny wynik, więc prawdopodobnie można by wyniki sporo ulepszyć, skupiając się bardziej na dobraniu parametrów. Jednak, co najważniejsze, otrzymane wyniki pokazują przy użyciu benchmarkowego zbioru instancji problemu, że podejście MCMC zastosowane wprost pozwala uzyskać wyniki $\sim 1.1 \cdot opt$ nawet bez dogłębnej analizy, co pokazuje jak dużą moc ma ta metoda, jeśli tylko odpowiednio zdefiniowany jest krok.

5.6 Podsumowanie rozdziału

Problem komiwojażera jest ważny w zastosowaniach, jako swego rodzaju *benchmarkowy* problem NP-trudny, dzięki któremu można ocenić skuteczność różnych metod aproksymacyjnych. (To jak ważny jest to problem np. w logistyce (wybór optymalnej trasy) jest oczywiste. W związku z tym został on dobrze zbadany i znanych jest wiele innych podejść do niego. Po pierwsze znany jest algorytm deterministyczny Helda-Karpa [25] o złożoności $O(n^2 2^n)$ – w praktyce ze względu na wykładniczą złożoność niemożliwy do zastosowania już dla średnio dużych n. Po drugie, znany jest wielomianowej złożoności algorytm Christofidesa $O(n^2 \log n)$ [26] dający suboptymalne rozwiązanie (co najwyżej $1,5$ razy dłuższa trasa niż optymalna) przy naturalnym założeniu nierówności trójkąta. Ponadto rozwinięte zostały różne aproksymacyjne podejścia probabilistyczne, takie jak: algorytmy mrówkowe [29], ewolucyjne, genetyczne [28] etc., znane są rozmaite heurystyki pozwalające na ulepszanie rozwiązań. Z punktu widzenia MCMC ciekawą propozycją jest *kwantowe wyżarzanie* [27] stosujące kwantowe metody MCMC na kwantowym komputerze (bazą zamiast fizyki statystycznej jest więc fizyka kwantowa) – mógłby to być temat na osobną pracę. Zastanowić się można nad łączeniem wymienionych tutaj metod z MCMC i zbadaniem, czy przynosi to poprawę wyników.

Dobre omówienie wielu metod, które można zastosować do problemu komiwojażera można znaleźć np. w [19]. Rozwiązania, które tu przedstawiłem, nie są nowe - są znane od lat. Jednak myślę, że udało się dobrze pokazać, że stosując klasyczne metody MCMC (które przy okazji mają świetną podbudowę teoretyczną w teorii łańcuchów Markowa), w prostym do implementacji algorytmie (bez nadmiernej troski o parametry), jesteśmy w stanie w krótkim czasie uzyskać wyniki niedużo gorsze od optymalnych.

6 O programie

Kod źródłowy jest publicznie dostępny i znajduje się w repozytorium: [39]

Jak wspomniałem we wstępie, program został napisany w języku Python3. Kod jest raczej tylko szkicem służącym prezentacji omawianych tu zagadnień za pomocą różnych symulacji, stąd nie zawsze użyte rozwiązania są najładniejsze. Np. kodowanie i dekodowanie za pomocą wszystkich szyfrów ma bardzo wiele wspólnych części, stąd dużo większą niż aktualnie część kodu można by uwspólnić – dla części niewspólnych stosując np. *factory pattern*. Ponadto, wiele elementów kodu wymagałoby uporządkowania. Niemniej jednak, co najważniejsze, program działa, więc spełnił swój cel.

6.1 Krótkie omówienie funkcjonalności i implementacji

Program podzieliłem na dwie główne części: w folderze *decryption_problem* znajdują się funkcjonalności i testy związane z problemem dekodowania zaszyfrowanego tekstu. W folderze *travelling_salesman_problem* zaś funkcjonalności związane z problemem komiwojażera.

6.1.1 Dekodowanie

W części *alphabetic* w pliku *alphabetic.py* znajdziemy funkcjonalności związane z wprowadzoną tutaj definicją alfabetu i tekstu.

```
class Alphabet(object):
```

jest klasą, która przechowuje wszystkie własności alfabetu: przede wszystkim listę jego *liter*, a także ich pozycje w alfabecie, co umożliwia użycie tej klasy do kodowania i dekodowania. Natomiast:

```
class StrippedText(object):
```

zadaje strukturę umożliwiającą szybką iterację po tekście, w którym część znaków pochodzi spoza alfabetu (patrz 4.4).
W folderze *cipher* znajdują się funkcjonalności pozwalające zakodować i odkodować tekst przy znajomości klucza dla szyfru Vigenère'a (*vigenere.py*), z autokluczem (*autokey.py*) i rozszerzonego (*vigenere_extended.py*). W przypadku klasycznego szyfru Vigenère'a i szyfru rozszerzonego za szyfrowanie i odszyfrowanie tekstu odpowiadają te same funkcje

```
def encrypt_decrypt_text(text, shift_key, alphabet):
```

Szyfr rozszerzony – dla wygody implementacji – przedstawiamy w postaci listy par $(i(a), b)$, gdzie szyfrujemy zgodnie z wzorem $ax + b \mod n_k$, gdzie $a \in \mathbb{Z}_{n_k}^*$, $i(a)$ jest indeksem a na liście liczb względnie pierwszych z n_k mniejszych od n_k. W związku z tym do procesu odszyfrowania potrzebujemy tej właśnie listy, tak więc funkcja ma postać:

```
def encrypt_decrypt_text(text, affine_key, alphabet, coprimes):
```

coprimes może zostać obliczone na samym początku, a potem przekazywane jako argument, dzięki czemu nie zwiększa to w żaden sposób złożoności. W przypadku autoklucza do szyfrowania i deszyfrowania potrzebujemy dwóch oddzielnych funkcji:

```
def encrypt_text(text, shift_key, alphabet):

def decrypt_text(text, shift_key, alphabet):
```

Każdy z szyfrujących/deszyfrujących pakietów dostarcza także funkcjonalności pomocniczych takich jak update dekodowania na danym indeksie (*update_decryption_by_key_index*) (patrz 4.4) czy podanie klucza dekodującego dla danego klucza kodującego (*reverse_key*). W pliku *substitution.py* znajduje się zapis prób z dekodowania złożenia szyfru podstawieniowego z szyfrem Vigenère'a – jak wspomniałem te próby nie były na razie udane.

W folderze *common* znajdują się pomocnicze funkcje wspólne dla wszystkich algorytmów dekodowania. Np. obliczanie częstości *n-gramów* w tekście:

```
def calculate_n_gram_frequencies(text, n):
```

zwracanie listy *n-gramów* związanych z i-tą pozycją szyfru:

```
def get_n_grams_with_i(text, n, i):
```

Obliczanie log-wagi i zmiany log-wagi (przy zadanym wyuczonym wcześniej nieznormalizowanym rozkładzie log-częstości – *log_distribution* oraz w drugim przypadku danym słowniku zmian częstości w tekście w porównaniu z aktualnym dekodowaniem):

```
def get_frequencies_change(old_frequencies, new_frequencies):
...
def calculate_log_weight_change(frequencies_change, log_distribution):
```

Itd. Ponadto znajduje się w części wspólnej funkcja:

```
def consistency(guessed_key, real_key, alphabet):
```

która pozwala wyliczyć na ilu procentowo polach w tekście dwa klucze są zgodne.
Te wspólne funkcje zostają następnie użyte przez implementacje algorytmów deszyfrujących, które znajdują się w folderze *algorithm*. Dla każdego algorytmu jest część *_calculator.py* odpowiadająca za *szybkie* obliczenie zmiany funkcji wagi, tak by można było zdecydować o następnym stanie w wędrówce MCMC:

```
def get_frequency_change_fixed_key_length(old_key, new_key,
                                          n_gram_length,
                                          current_decryption,
                                          alphabet):
```

new_key jest przedstawiany jako krotka (i, j), gdzie i jest pozycją, na której zaszła zmiana w porównaniu ze starym kluczem, a j jest przesunięciem w porównaniu do starego klucza (czyli gdy 1 zmienia się na 2, to $j = 1$). W przypadku klucza rozszerzonego przedstawienie to jest po prostu w postaci (i, j), gdzie j jest nowym kluczem pojedynczym na i-tej pozycji (tam trzeba również podać listę liczb względnie pierwszych z długością alfabetu). W pakietach *_neighbours* są zawarte funkcjonalności generowania kandydata. Generalnie polega to na tym, że numerujemy wszystkich sąsiadów stanu od 0 do $N - 1$, gdzie N jest liczbą sąsiadów i losujemy jednostajnie liczbę od 0 do $N - 1$ – najnowszym kandydatem jest ten związany z wylosowaną liczbą. Równie dobrze można by też losować najpierw pozycję, a potem zmianę na tej pozycji (teoretycznie losowanie tylko raz pozwala nam zachować więcej losowości na później, jednak w praktyce generator z modułu *random* ma tak długi cykl, że nie ma to znaczenia).

Ponieważ przestrzenie szyfrów z autokluczem i Vigenère'a przedstawiają się w ten sam sposób – mają też wspólne pakiety generujące kandydatów (sąsiadów). Funkcja generująca ma postać:

```
def get_candidate(current, alphabet)
```

Gdzie *current* jest aktualnym stanem (w przypadku klucza rozszerzonego znów podajemy jeszcze listę liczb względnie pierwszych – dla oszczędności czasu).
Ostatnim elementem są właściwe pakiety dekodujące *_decoder.py*. Znajdują się tam dla każdego typu szyfru cztery funkcje:

```
def get_max_monogram_state(encryption, monogram_log_distribution, key_length,
alphabet):
...
def get_max_bigram_state(encryption, bigram_log_distribution, key_length, alphabet):
...
def break_fixed_length_code_with_mcmc
(encryption,
alphabet,
starting_state,
n_list,
coefs,
log_distributions,
steps,
true_decrypting_code=[],
consistency_thresholds=[])
```

```
...
def break_bounded_length_code_with_mcmc_monogram_criteria
(encryption,
alphabet,
n_list,
coefs,
log_distributions,
steps_constant,
lower_bound, upper_bound,
monogram_log_distribution,
scrabble):
...
```

przy czym w pewnych wypadkach dla szyfru *rozszerzonego* może występować dodatkowy argument – lista liczb względnie pierwszych z dł. alfabetu. Pierwsza z funkcji deterministycznie (patrz 4.4) zwraca monogramowy *argmax* (i jego log-wagę), druga robi to samo dla bigramowego.
Trzecia z funkcji implementuje *wędrówkę MCMC* dla problemu dekodowania szyfrogramu (*encryption*) – dla listy *n-gramów* branych pod uwagę przy liczeniu log-wagi (*n_list* i współczynników przy każdym z nich *coefs*, w naszych symulacjach listy n są jednoelementowe (postaci (2) dla bigramów lub (3) dla trigramów), natomiast *coefs* to po prostu (1.0) (patrz 4.3). *log_distributions* to lista słowników log-częstości w języku angielskim dla wszystkich n-gramów z *n_list* (w naszym wypadku tylko dla jednego). *steps* to liczba kroków – jak długo ma trwać wędrówka. Dodatkowo można podać *true_decrypting_code* – nominalnie prawdziwy klucz deszyfrujący (w praktyce może to być coś innego, np. *argmax* bigramowy) i *consistency_thresholds* – progi zgodności do osiągnięcia przez łańcuch z *true_decrypting_code*. Gdy osiągnięty zostaje ostatni próg, łańcuch kończy wędrówkę. Jest to funkcjonalność wprowadzona dodatkowo dla celów testowych. Funkcja zwraca ostatecznie odgadnięty klucz deszyfrujący (najlepszy odwiedzony stan), jego log-wagę, dodatkowo może zwracać listę kroków, w których osiągane były kolejne progi. Dla szyfru Vigenère'a dodatkowo zwracany jest krok, w którym został osiągnięty najlepszy stan (również dla celów testowych).
Ostatnia funkcja jest implementacją algorytmu dekodującego klucz o nieznanej acz ograniczonej długości. *steps_constant* odpowiada stałej C z wcześniejszego rozdziału *lower* i *upper bound* to ograniczenia długości klucza. Są poza tym podane parametry do wędrówki na ustalonej długości kluczach oraz słownik log-częstości monogramowych do wyliczania *argmax* monogramowego. Na końcu podaje się *scrabble* – zbiór słów w języku angielskim.

W folderze *data* znajdują się funkcje generujące słowniki i teksty źródłowe, a także wygenerowane słowniki, w folderze *testing_and_results* są zaimplementowane opisane w tej pracy testy (jest tam też krótki opis przed każdym testowanym przypadkiem), a także zebrane ich wyniki.

Jeśli dla danej części kodu zostały napisane testy jednostkowe, to znajdują się one w podfolderach *unit_tests*.

6.1.2 Problem komiwojażera

Struktura jest tu dużo prostsza. Mamy trzy foldery *algorithm*, *data* oraz *testing_and_results*.
W pierwszym z nich znajduje się implementacja algorytmu. W *neighbours.py* generowane są losowe *swapy* na permutacjach:

```
def get_random_swap(size):
```

size to rozmiar problemu (liczba wierzchołków) – nic więcej nie potrzeba, żeby losowy *swap* wygenerować (następnie używany on jest do wygenerowania kandydata albo zgodnie z opcją 1 albo 2 z rozdziału o problemie komiwojażera). Ponadto losujemy tam też stan startowy:

```
def get_random_starting_state(size):
```

W *calculator.py* znajdują się funkcje obliczające *update'y* funkcji stanu (sumy odległości) dla potencjalnego przejścia do zaproponowanego kandydata dla opcji 1 i 2:

```
def get_state_function_update1(distances, current_state, swap):
def get_state_function_update2(distances, current_state, swap):
```

Oraz funkcje wykonujące ten update:

```
def update_state(current_state, swap):
...
def update_state_reverse_swap(current_state, swap):
...
```

Ostatecznie te wszystkie funkcje są użyte w *solver.py*, który implementuje podejście MCMC do problemu:

```
def solve_max_steps1(filename_or_distances, steps):
...
def solve_convergence1(filename_or_distances, c):
...
def solve_max_steps2(filename_or_distances, steps):
...
def solve_convergence2(filename_or_distances, c):
...
def solve3(filename_or_distances, steps)
```

Liczba oznacza użytą opcję macierzy przejścia, *filename_or_distances* to albo adres pliku *.xml* z macierzą odległości w postaci takiej jak w TSPLIB, albo już wygenerowana macierz odległości w postaci słownika. Wersje *max_steps* to JŁM ($t_n = 1$) z ograniczeniem liczby kroków przez *steps*. Wersje *convergence* to wersje *zbieżnościowe* z wyżarzaniem $t_n = \frac{c}{\log(n+2)}$ i kryterium zbieżności jak opisane we wcześniejszym rozdziale. W wartości zwracanej dostajemy informacje, które zostały użyte w (5.5). Jak pamiętamy, przyjmowaliśmy $c = 3$. Widać, że łatwo można by było zmniejszyć objętość kodu np. podając numer opcji jako argument: tak jak wspominałem – w wielu miejscach rozwiązania mogłyby być ładniejsze. *solve3* to funkcja odpowiadająca za losowanie jednostajne rozwiązania dla zbioru wejściowego *filename_or_distances* tyle razy, ile określa to *steps*.
W folderze *data* znajdują się funkcje generujące macierze odległości na podstawie plików *.xml*, natomiast w folderze *testing_and_results* znajdują się zapis i wyniki przeprowadzonych testów.

Pliki .xml nie są zawarte w repozytorium, zajmowały bowiem za dużo miejsca. Łatwo natomiast można je znaleźć na stronie TSPLIB [4].

6.2 Uwagi

W niektórych miejscach – poza kwestiami estetycznymi – dałoby się program nieco zoptymalizować i przyspieszyć w ten sposób jego działanie. Podobnie można by użyć *szybszego* kompilowanego, a nie interpretowanego języka.
Niemniej jednak byłaby to optymalizacja o stałą, a program w wersji obecnej, co najważniejsze, realizuje zagadnienia przedstawione w tej pracy, z takimi złożonościami czasową i pamięciową, jakie zostały opisane w pracy.

7 Podsumowanie

W swojej pracy najpierw przypomniałem najważniejsze własności łańcuchów Markowa. Starałem się przeprowadzać dowody wg własnego toku rozumowania - choć oczywiście inspirowałem się m.in. [6] czy wykładami jak [21]. Przedstawione własności łańcuchów Markowa - przede wszystkim twierdzenie ergodyczne - posłużyły następnie jako uzasadnienie dla zastosowania algorytmów Metropolisa i Metropolisa-Hastingsa.
W części dotyczącej szyfrowania udało mi się stworzyć i zaimplementować algorytm deterministyczny oparty na programowaniu dynamicznym rozwiązujący postawioną tu wersję problemu dekodowania w wielomianowym czasie. Następnie pokazałem, jak - nie tracąc na jakości - przyspieszyć uzyskanie rozwiązania przy pomocy nowej metody używającej w odpowiedni sposób MCMC. Działanie algorytmów zaprezentowałem na różnych szyfrach i alfabetach. Pokazałem też potencjalne przyszłe kierunki, w jakich można by rozwinąć zaprezentowane podejście.
W końcowej części przedstawiłem pewne intuicje dotyczące problemu komiwojażera, a następnie zaprezentowałem próbę rozwiązania wybranych instancji tego problemu z TSPLIB za pomocą metod MCMC (m.in. symulowanego wyżarzania), uzyskując wyniki ok. 1.1 razy gorsze od optymalnych.

Całość pracy pokazuje, że przy dobrej konstrukcji łańcucha metody MCMC mogą być skutecznie zastosowane do rozwiązania omawianych problemów.

Literatura

[1] P. Diaconis, *The Markov Chain Monte Carlo Revolution* Bull. Amer. Math. Soc., Nov. 2008.

[2] S. Connor, *Simulation and Solving Substitution Codes.* Master's thesis. Department of Statistics, University of Warwick 2003

[3] Chen, J., Rosenthal J.S. *Decrypting classical cipher text using Markov chain Monte Carlo.* Stat Comput 22, 397–413 2012.

[4] TSPLIB – http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, [dostęp: 03.09.2020]

[5] Tożsamość Bézouta, https://en.wikipedia.org/wiki/Bezout's_identity, [dostęp: 03.09.2020]

[6] J.Jakubowski, R.Sztencel, *Wstęp do teorii prawdopodobieństwa.* Wydawnictwo SCRIPT Wydanie: 4 rozszerz., 2010

[7] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, *Equations of state calculations by fast computing machines* J. Chem. Phys. 21 1953, 1087.

[8] W. K. Hastings, *Monte carlo sampling methods using markov chains and their applications*, Biometrika 57 1970, no. 1, 97.

[9] P. H. Peskun, *Optimum Monte-Carlo Sampling Using Markov Chains*, Biometrika, Vol. 60, No. 3. 1973, pp. 607-612

[10] Szyfr Cezara - Wiki: https://en.wikipedia.org/wiki/Caesar_cipher, [dostęp: 03.09.2020]

[11] Szyfr Vigenère'a - Wiki: https://en.wikipedia.org/wiki/Vigenere_cipher https://pl.wikipedia.org/wiki/Szyfr_Vigenere'a, [dostęp: 03.09.2020]

[12] Szyfr z autokluczem - Wiki: https://en.wikipedia.org/wiki/Autokey_cipher, [dostęp: 03.09.2020]

[13] F. W. Kasiski, *Die Geheimschriften und die Dechiffrir-Kunst* E. S. Mittler und Sohn, 1863.

[14] W. Friedman, *The index of coincidence and its applications in cryptanalysis*, 1987.

[15] G. A. Croes, *A method for solving traveling salesman problems.* Operations Res. 6 1951, pp., 791-812.

[16] M. M. Flood, *The traveling-salesman problem. Operations* Res. 4 1956, pp., 61-75.

[17] K.I. Rahmani et al. *Alpha-qwerty cipher: an extended Vigenère cipher*, Advanced Computing: An International Journal, May 2012,

[18] A.A. Soofi, I. Riaz, U. Rasheed, *An Enhanced Vigenère Cipher For Data Security*, International Journal of Scientific & Technology Research Volume 5, Issue 03, March 2016

[19] David S. Johnson1, Lyle A. McGeoch2, *The Traveling Salesman Problem: A Case Study in Local Optimization*, November 20, 1995

[20] D, A. Levin, Y, Peres, E. L. Wilmer, *Markov Chains and Mixing Times*, University of Oregon Microsoft Research, University of California, Berkeley, Oberlin College, June 3, 2007

[21] Krótki wykład na temat *couplingu* i twierdzenia ergodycznego https://www2.cs.duke.edu/courses/spring13/compsci590.2/slides/lec5.pdf, [dostęp: 03.09.2020]

[22] A. Bhateja et.al, *Cryptanalysis of Vigenère Cipher using Particle Swarm Optimization with Markov chain random walk*, Vol. 5 No. 05 May 2013

[23] A.J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, IEEE Transactions on Information Theory. 13 (2): 260–269, April 1967

[24] R. Bellman, *Dynamic programming treatment of the travelling salesman problem*, Journal of Assoc. Computing Mach. 9. 1962.

[25] M.Held, R. M. Karp, *A dynamic programming approach to sequencing problems*, Journal for the Society for Industrial and Applied Mathematics 1:10. 1962

[26] N. Christofides, *Worst-case analysis of a new heuristic for the travelling salesman problem*, Report 388, Graduate School of Industrial Administration, CMU, 1976.

[27] E. Crosson, A. W. Harrow, *Simulated Quantum Annealing Can Be Exponentially Faster Than Classical Simulated Annealing*, 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)

[28] N. M. Razali, J. Geraghty *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*, Proceedings of the World Congress on Engineering 2011 Vol II

[29] M.Dorigo, L. M. Gambardella *Ant colonies for the travelling salesman problem*, Biosystems Volume 43, Issue 2, July 1997, Pages 73-81

[30] *Przykładowy słownik częstości n-gramów:* http://practicalcryptography.com/cryptanalysis/text-characterisation/monogram-bigram-and-trigram-frequency-counts/, [dostęp: 03.09.2020]

[31] *Gutenberg project* https://www.gutenberg.org/, [dostęp: 03.09.2020]

[32] C. E. Shannon, *Communication Theory of Secrecy Systems, Bell Systems Technical Journal*, Vol. 28, pp. 656–715, 1948

[33] B. Hajek, *Cooling schedules for optimal annealing*, Mathematics of Operation Research, Vol. 13. No. 2. May 1988

[34] https://en.wikipedia.org/wiki/Index_of_coincidence, [dostęp: 03.09.2020]

[35] *Open-source'owy słownik scrabblisty* https://www.wordgamedictionary.com/sowpods/download/sowpods.txt

[36] *Hokej na lodzie - Wiki:* https://en.wikipedia.org/wiki/Ice_hockey, [dostęp: 03.09.2020]

[37] G. Reinelt, *TSPLIB—A Traveling Salesman Problem Library*, Journal on Computing, 1991, vol. 3, issue 4, 376-384

[38] D. Applegate et al., *Certification of an optimal TSP tour through 85,900 cities*, Operations Research Letters 37(1), 2009

[39] Cyryl Karpiński, *Repozytorium z kodem źródłowym*, https://github.com/cyrylk/MCMC/