Adrianna Rapa 173204
Natalia Rozner 173205
Inżynieria i Analiza Danych

Model Isinga i Model Pottsa-Dirichleta

Projekt z przedmiotu

Administracja Systemów Rozproszonych

Opiekun pracy:
mgr inż. Dariusz Rączka

Rzeszów, 2024

Spis treści

Spis wykresów:

Spis zrzutów ekranu kodu źródłowego:

Spis równań:

Wykaz symboli, oznaczeń i skrótów

Funkcje:

- **plot_spin_lattice** - Funkcja do rysowania siatki spinów i wykresu słupkowego.
- **plot_spin_matrix** - Funkcja do rysowania macierzy spinów w postaci siatki 2D.
- **calculate_energy** - Funkcja obliczająca energię układu spinów w modelu Isinga.
- **metropolis** - Algorytm Metropolisa do symulacji modelu Isinga.
- **simulate_ising_and_potts** - Funkcja do symulacji modeli Isinga i Potts-Dirichleta oraz wizualizacji zmian w siatkach i wykresach.
- **analyze_phase_transitions** - Funkcja analizująca przejścia fazowe dla różnych rozmiarów siatki w modelu Isinga.
- **find_clusters** - Funkcja znajdująca klastry spinów.
- **plot_cluster_size_distribution** - Funkcja badająca rozkład rozmiarów klastrów spinów.
- **plot_3d_energy_surface** - Funkcja rysująca powierzchnię trójwymiarową energii w zależności od kroków Monte Carlo i temperatury.

Parametry:

- **N** - Rozmiar siatki.
- **i, j** - Indeksy dla wiersza i kolumny macierzy.
- **di, dj** - Przesunięcie w pionie i poziomie do sąsiadów.
- **dE** - Zmiana energii w procesie Monte Carlo.
- **q** - Parametr q w modelu Potts-Dirichleta (wartości, jakie mogą przyjmować spiny).
- **J** - Stała sprzężenia magnetycznego w modelu Isinga.
- **kT** - Iloczyn stałej Boltzmanna i temperatury.

Zmienne:

- **size** - Rozmiar siatki.
- **temperature** - Temperatura modelu.
- **spins** - Macierz spinów.
- **energy** - Energia układu.
- **magnetization** - Magnetyzacja układu.
- **old_spin, new_spin** - Stare i nowe wartości spinu.
- **energies** - Lista zawierająca wartości energii w kolejnych krokach i temperaturach.
- **steps** - Liczba kroków Monte Carlo.
- **temperatures** - Lista temperatur.
- **X, Y, Z** - Współrzędne dla wykresu trójwymiarowej powierzchni energii.
- **clusters** - Lista przechowująca klastry spinów.
- **cluster_sizes** - Lista przechowująca rozmiary klastrów.

Klasy:

- **IsingModel** - Klasa reprezentująca model Isinga.
- **PottsDirichletModel** - Klasa reprezentująca model Potts-Dirichleta.

Pakiety:

- **np** - Pakiet NumPy.
- **plt** - Pakiet matplotlib.pyplot.

1. Wstęp

Niniejsza praca koncentruje się na analizie i porównaniu dwóch modeli fizycznych stosowanych w badaniach nad systemami rozproszonymi: modelem Isinga oraz modelem Potts-Dirichleta. Problematyka ta jest istotna z perspektywy administracji systemami rozproszonymi, gdzie analiza dynamiki układów wielociałowych odgrywa kluczową rolę w zrozumieniu ich zachowania.

Model Isinga, będący podstawowym modelem opisującym zachowanie magnetyczne w układach wieloelementowych, stanowi fundament w badaniach nad fenomenami fazowymi i przejściami fazowymi. Jego uproszczona struktura pozwala na analizę zjawisk, takich jak spontaniczne uporządkowanie magnetyczne, w kontekście systemów rozproszonych.

Z kolei model Potts-Dirichleta, będący uogólnieniem modelu Isinga, pozwala na badanie wielu stanów magnetycznych w układach o skomplikowanej dynamice. Jego zastosowanie w analizie systemów rozproszonych pozwala na uwzględnienie bardziej złożonych wzorców i struktur występujących w tych systemach.

Wybór tematyki badawczej wynika z potrzeby zrozumienia dynamiki i zachowania systemów rozproszonych w kontekście ich modelowania oraz analizy. Celem pracy jest przeprowadzenie porównawczej analizy modeli Isinga i Potts-Dirichleta pod kątem ich przydatności w kontekście administracji systemami rozproszonymi.

Zakres pracy obejmuje implementację obu modeli w języku Python, analizę ich zachowania przy różnych warunkach początkowych oraz interpretację uzyskanych wyników pod kątem ich przydatności w kontekście administracji systemami rozproszonymi. Przyjęte założenia obejmują uproszczenia związane z modelem Isinga oraz uwzględnienie wielu stanów magnetycznych w modelu Potts-Dirichleta.

Ostatecznym celem pracy jest zidentyfikowanie różnic między modelem Isinga a modelem Potts-Dirichleta oraz określenie ich przydatności w analizie i modelowaniu systemów rozproszonych. Poprzez przeprowadzenie eksperymentów numerycznych i analizę uzyskanych wyników, pragniemy wnosić nowe spojrzenie na rolę tych modeli w administracji systemami rozproszonymi.

2. Założenia projektu

Projekt powinien spełniać następujące wymagania:

- Model Isinga i Pottsa: Projekt zakłada analizę dwóch modeli fizycznych - modelu Isinga oraz modelu Pottsa - w kontekście ich zastosowania w administracji systemami rozproszonymi.
- Implementacja kodu: Projekt obejmuje implementację kodu komputerowego realizującego modele Isinga i Pottsa w języku Python.
- Analiza porównawcza: Praca ma na celu przeprowadzenie analizy porównawczej obu modeli w kontekście ich skuteczności i przydatności w administracji systemami rozproszonymi.
- Podstawy teoretyczne: Projekt uwzględnia omówienie podstaw teoretycznych obu modeli, aby czytelnik miał pełne zrozumienie ich działania i zastosowań.
- Symulacje i wyniki: Planowane są symulacje zachowania obu modeli w różnych warunkach środowiskowych, a uzyskane wyniki zostaną poddane analizie i interpretacji.
- Inicjalizacja parametrów: Projekt zakłada, że użytkownik będzie mógł podać parametry modelu (rozmiar siatki, temperatura, parametr q) na etapie inicjalizacji symulacji.
- Analiza przejść fazowych: Projekt obejmie analizę przejść fazowych w modelu Isinga w zależności od temperatury i rozmiaru siatki.
- Badanie rozkładu klastrów spinów: Praca uwzględnia również badanie rozkładu klastrów spinów w modelu Isinga w różnych temperaturach.

3. Opis teoretyczny

Model Isinga jest wykorzystywany do badania zachowania układów magnetycznych, takich jak ferromagnetyki. Każdy punkt w siatce jest opisany przez spin, który może przyjmować wartość -1 lub 1, reprezentując skierowanie magnetyzacji. Stan całego układu jest zależny od stanów poszczególnych spinów, a zmiana spinów może prowadzić do zmiany stanu całego układu.

Energia całego systemu, czyli Hamiltonian, jest określana przez następujące równanie:

$$H = -J \sum_{<i,j>} s_i s_j - H_0 \sum_i \delta\,(S_i \alpha)$$

Równanie 1 - Hamiltonian dla modelu Isinga

gdzie:

- J - to stała opisująca siłę oddziaływań między sąsiednimi spinami w siatce
- Σ<i,j> - suma po sąsiednich parach spinów w siatce
- s_i - to stan punktu na i-tym miejscu siatki (przyjmuje wartości -1 lub 1)
- $\delta(s_i, s_j)$ - to delta Kroneckera, równa 1, jeśli $s_i = \alpha$, lub 0 w przeciwnym przypadku
- H_0 - stała opisująca wpływ zewnętrznego pola magnetycznego na układ
- α - to stan, który jest faworyzowany przez zewnętrzne pole magnetyczne.

Pierwszy składnik Hamiltonianu reprezentuje energię wynikającą z oddziaływań między sąsiednimi spinami o tym samym skierowaniu magnetyzacji. Im bardziej spiny są skierowane w tym samym kierunku, tym silniejsze jest oddziaływanie. Drugi składnik opisuje wpływ zewnętrznego pola magnetycznego, które faworyzuje określony stan α.

W projekcie analizujemy wpływ zewnętrznego pola magnetycznego H_0 na symulację.

Model Potts-Dirichleta jest rozszerzeniem modelu Isinga, pozwalającym na więcej niż dwa możliwe stany spinów. Każdy punkt w siatce może przyjąć wartość od 1 do q, gdzie q to liczba stanów możliwych dla spinu.

Energia całego systemu jest opisana przez Hamiltonian:

$$H = \sum_{i<j} J_{ij}(s_i s_j) + \sum_i h_i\,(s_i)$$

Równanie 2 - Hamiltonian dla modelu Potts-Dirichleta

gdzie:

- pierwsza suma oznacza oddziaływanie między parami spinów,
- a druga suma opisuje jednoczesne oddziaływanie spinów z zewnętrznym polem magnetycznym

W projekcie badamy wpływ oddziaływania sumy $h_i(s_i)$ na symulację.

4. Opis działania algorytmu w krokach

4.1. Definicja funkcji i klas:

- Funkcje do rysowania:
    - plot_spin_lattice: Rysuje siatkę spinów w postaci graficznej.
    - plot_spin_matrix: Rysuje macierz spinów w postaci siatki 2D.
- Klasy modeli:
    - IsingModel: Reprezentuje model Isinga.
    - PottsDirichletModel: Reprezentuje model Potts-Dirichleta.

4.2. Implementacja algorytmów:

- Algorytm Metropolisa:
    - metropolis: Implementuje algorytm Metropolisa do symulacji modelu Isinga.
- Symulacja Monte Carlo:
    - Metody monte_carlo_step w klasach modeli wykonują symulację Monte Carlo dla odpowiednich modeli.

4.3. Symulacja i wizualizacja wyników:

- Funkcja simulate_ising_and_potts:
    - Przeprowadza symulację modelu Isinga i modelu Potts-Dirichleta dla zadanych parametrów.
    - Zapisuje energie i magnetyzacje po każdym kroku symulacji.
    - Wyświetla wyniki w postaci siatek przed i po zmianach oraz wykresów zmiany energii i magnetyzacji.
- Obsługa wejścia użytkownika:
    - Pobiera rozmiar siatki, temperaturę i parametr q od użytkownika.
    - Wywołuje funkcję simulate_ising_and_potts z wczytanymi parametrami.

4.4. Dodatkowe funkcje analizy i wizualizacji:

- Analiza przejść fazowych:
    - Funkcja analyze_phase_transitions analizuje przejścia fazowe dla różnych rozmiarów siatki.
- Rozkład rozmiarów klastrów spinów:
    - Funkcja plot_cluster_size_distribution bada rozkład rozmiarów klastrów spinów.

4.5. Implementacja sugestii:

- Dodatkowe funkcje analizy:
    - find_clusters: Znajduje klastry spinów.
    - plot_3d_energy_surface: Rysuje powierzchnię trójwymiarową energii w zależności od kroków i temperatury.

5. Opis działania kodu krok po kroku

- Użyte biblioteki:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

Zrzut ekranu kodu źródłowego 1

5.1. Funkcja 'plot_spin_lattice(spins, title, cmap='binary')'

- Opis funkcji: Funkcja służy do rysowania siatki spinów wraz z wykresem słupkowym prezentującym rozkład liczności poszczególnych wartości spinów

- Parametry:
  spins: Macierz zawierająca wartości spinów.
  title: Tytuł wykresu.
  cmap='binary': Parametr opcjonalny określający mapę kolorów. Domyślnie ustawiony na 'binary', co odpowiada modelowi Isinga.

- Kroki działania:
  1. Inicjalizacja nowego wykresu o rozmiarach 8x8.
  2. Wyświetlenie siatki spinów na wykresie z wykorzystaniem odpowiedniej mapy kolorów.
  3. Znalezienie unikalnych wartości spinów oraz ich liczności.
  4. Określenie kolorów dla poszczególnych wartości spinów w zależności od wybranej mapy kolorów:
  5. Dla mapy 'binary' (model Isinga): wartości spinów -1 oznaczane są kolorem szarym, a 1 - kolorem czarnym.
  6. Dla innych map kolorów (model Potts-Dirichleta): losowane są kolory z palety HSV dla każdej wartości spinu.
  7. Rysowanie wykresu słupkowego, gdzie wysokość słupków odpowiada liczbie wystąpień poszczególnych wartości spinów.
  8. Ustawienie etykiet osi y na podstawie wartości spinów.
  9. Odwrócenie osi y, aby etykiety były od dołu do góry.
  10. Dodanie etykiet osi x, osi y oraz tytułu wykresu.
  11. Wyświetlenie wykresu.

```python
def plot_spin_lattice(spins, title, cmap='binary'):
    """Funkcja do rysowania siatki i wykresu słupkowego."""
    plt.figure(figsize=(8, 8))

    # Wyświetlanie siatki spinów
    plt.imshow(spins, cmap=cmap, interpolation='nearest')
    plt.title(title)
    plt.axis('off')
    plt.show()

    # Znalezienie unikalnych wartości spinów i ich liczności
    unique, counts = np.unique(spins, return_counts=True)


    if cmap == 'binary':
        # Dla modelu Isinga
        colors = {-1: 'grey', 1: 'black'}
    else:
        # Dla modelu Potts-Dirichleta
        # Losowanie kolorów dla każdej wartości spinu i zapisanie ich do słownika
        hsv_colors = plt.cm.hsv(np.linspace(0, 1, len(unique)))
        colors = dict(zip(unique, hsv_colors))

    # Rysowanie wykresu słupkowego odpowiadającego liczbie wystąpień danej wartości spinu
    for i, (spin, count) in enumerate(zip(unique, counts)):
        color = colors[spin] if cmap != 'binary' else colors.get(spin, 'grey')
        plt.barh(i, count, color=color)

    # Ustawienie etykiet osi y i ich wartości
    plt.yticks(range(len(unique)), unique)
    plt.gca().invert_yaxis()  # Odwrócenie osi y, aby etykiety były od dołu do góry
    plt.xlabel('Liczba wystąpień')
    plt.ylabel('Wartość spinu')
    plt.title('Rozkład spinów')
    plt.show()
```

Zrzut ekranu kodu źródłowego 2

5.2. Klasa ‘IsingModel’

- Opis klasy: Reprezentuje model Isinga.

5.2.1. Metoda __init__(self, size, temperature):

- Parametry:
  size: Rozmiar siatki.
  temperature: Temperatura układu.

- Kroki działania:
  1. Inicjalizacja rozmiaru siatki i temperatury.
  2. Losowe wygenerowanie spinów o rozmiarze size x size.

5.2.2. Metoda energy(self):

- Kroki działania:
  1. Obliczenie energii układu na podstawie sumy produktów sąsiadujących spinów.

5.2.3. Metoda magnetization(self):

- Kroki działania:
  1. Obliczenie sumy wszystkich spinów, co odpowiada magnetyzacji układu.

5.2.4. Metoda monte_carlo_step(self):

- Kroki działania:
  1. Losowy wybór współrzędnych (i, j) w siatce.
  2. Obliczenie zmiany energii po ewentualnej zmianie spinu.
  3. Zmiana spinu zgodnie z prawdopodobieństwem zgodnie z regułą Metropolisa.

```
class IsingModel:
    """Klasa reprezentująca model Isinga."""

    def __init__(self, size, temperature):
        """Inicjalizacja modelu."""
        self.size = size
        self.temperature = temperature
        self.spins = np.random.choice([-1, 1], size=(size, size))

    def energy(self):
        """Obliczenie energii."""
        return -np.sum(self.spins * (np.roll(self.spins, 1, axis=0) + np.roll(self.spins, 1, axis=1)))

    def magnetization(self):
        """Obliczenie magnetyzacji."""
        return np.sum(self.spins)

    def monte_carlo_step(self):
        """Jeden krok Monte Carlo."""
        i, j = np.random.randint(0, self.size, 2)
        dE = 2 * self.spins[i, j] * (self.spins[(i + 1) % self.size, j] + self.spins[(i - 1) % self.size, j] +
                                     self.spins[i, (j + 1) % self.size] + self.spins[i, (j - 1) % self.size])
        if dE <= 0 or np.random.rand() < np.exp(-dE / self.temperature):
            self.spins[i, j] *= -1
```

Zrzut ekranu kodu źródłowego 3

5.3. Klasa PottsDirichletModel

- Opis klasy: Reprezentuje model Potts-Dirichleta.

5.3.1. Metoda __init__(self, size, temperature, q):

- Parametry:

  size: Rozmiar siatki.

  temperature: Temperatura układu.

  q: Parametr określający ilość możliwych stanów spinu.

- Kroki działania:
  1. Inicjalizacja rozmiaru siatki, temperatury i parametru q.
  2. Losowe wygenerowanie spinów o rozmiarze size x size.

5.3.2. Metoda energy(self):

- Kroki działania:
  1. Obliczenie energii układu na podstawie reguł dla modelu Potts-Dirichleta.

5.3.3. Metoda magnetization(self):

- Kroki działania:
  1. Obliczenie sumy wszystkich spinów, co odpowiada magnetyzacji układu.

5.3.4. Metoda monte_carlo_step(self):

- Kroki działania:
  1. Losowy wybór współrzędnych (i, j) w siatce.
  2. Wybór nowego spinu różnego od aktualnego.
  3. Obliczenie zmiany energii po ewentualnej zmianie spinu.
  4. Zmiana spinu zgodnie z prawdopodobieństwem zgodnie z regułą Metropolisa.

```python
class PottsDirichletModel:
    """Klasa reprezentująca model Potts-Dirichleta."""

    def __init__(self, size, temperature, q):
        """Inicjalizacja modelu."""
        self.size = size
        self.temperature = temperature
        self.q = q
        self.spins = np.random.randint(1, q + 1, size=(size, size))

    def energy(self):
        """Obliczenie energii."""
        energy = 0
        for i in range(self.size):
            for j in range(self.size):
                energy -= sum(
                    self.spins[i, j] == self.spins[(i + di) % self.size, (j + dj) % self.size] for di in [-1, 1] for dj
                    in [-1, 1])
        return energy

    def magnetization(self):
        """Obliczenie magnetyzacji."""
        return np.sum(self.spins)

    def monte_carlo_step(self):
        """Jeden krok Monte Carlo."""
        i, j = np.random.randint(0, self.size, 2)
        old_spin = self.spins[i, j]
        new_spin = old_spin
        while new_spin == old_spin:
            new_spin = np.random.randint(1, self.q + 1)
        dE = sum(new_spin == self.spins[(i + di) % self.size, (j + dj) % self.size] -
                 old_spin == self.spins[(i + di) % self.size, (j + dj) % self.size] for di in [-1, 1] for dj in [-1, 1])
        if dE <= 0 or np.random.rand() < np.exp(-dE / self.temperature):
            self.spins[i, j] = new_spin
```

Zrzut ekranu kodu źródłowego 4

5.4. Funkcja plot_spin_matrix(spins, title)

- Opis funkcji: Rysuje macierz spinów w formie siatki 2D.

- Parametry:
  spins: Macierz spinów.
  title: Tytuł wykresu.

- Kroki działania:
  1. Rysowanie siatki spinów na wykresie w formie szarości.
  2. Dodanie tytułu do wykresu.
  3. Wyświetlenie wykresu.

```
def plot_spin_matrix(spins, title):
    """Funkcja rysuje macierz spinów w postaci siatki 2D."""
    plt.imshow(spins, cmap='gray', vmin=-1, vmax=1)
    plt.title(title)
    plt.show()
```

Zrzut ekranu kodu źródłowego 5

5.5. Funkcja calculate_energy(spins, J)

- Opis funkcji: Oblicza energię układu spinów w modelu Isinga.

- Parametry:
  spins: Macierz spinów.
  J: Stała sprzężenia magnetycznego.

- Kroki działania:
  1. Inicjalizacja energii na 0.
  2. Iteracja po wszystkich spinach, obliczenie sumy ich oddziaływań i dodanie do energii.
  3. Zwrócenie obliczonej energii.

```
def calculate_energy(spins, J):
    """
    Funkcja calculate_energy(spins, J) oblicza energię układu spinów
    w modelu Isinga.
    spins: macierz spinów
    J: stała sprzężenia magnetycznego
    """
    energy = 0
    N = spins.shape[0]
    for i in range(N):
        for j in range(N):
            energy += -J * spins[i, j] * (spins[(i + 1) % N, j] + spins[i, (j + 1) % N])
    return energy
```

5.6. Funkcja metropolis(spins, J, kT, num_steps)

- Opis funkcji: Implementuje algorytm Metropolisa do symulacji modelu Isinga.

- Parametry:
  spins: Macierz spinów.
  J: Stała sprzężenia magnetycznego.
  kT: Iloczyn stałej Boltzmanna i temperatury.
  num_steps: Liczba kroków symulacji.

- Kroki działania:
  1. Iteracja po zadanym liczbie kroków.
  2. Losowy wybór spinu.
  3. Obliczenie zmiany energii po ewentualnej zmianie spinu.
  4. Zmiana spinu zgodnie z prawdopodobieństwem zgodnie z regułą Metropolisa.
  5. Zwrócenie zmodyfikowanej siatki spinów i listy energii.

```python
def metropolis(spins, J, kT, num_steps):
    """
    Algorytm Metropolisa do symulacji modelu Isinga.
    spins: macierz spinów
    J: stała sprzężenia magnetycznego
    kT: iloczyn stałej Boltzmanna i temperatury
    num_steps: liczba kroków symulacji
    """
    N = spins.shape[0]
    energies = []
    for step in range(num_steps):
        i = np.random.randint(0, N)
        j = np.random.randint(0, N)
        spin_flip_energy = 2 * J * spins[i, j] * (spins[(i - 1) % N, j] + spins[(i + 1) % N, j] +
                                                  spins[i, (j - 1) % N] + spins[i, (j + 1) % N])
        if spin_flip_energy <= 0 or np.exp(-spin_flip_energy / kT) > np.random.rand():
            spins[i, j] *= -1
        energy = calculate_energy(spins, J)
        energies.append(energy)
    return spins, energies
```

Zrzut ekranu kodu źródłowego 6

5.7. Funkcja simulate_ising_and_potts

- Opis funkcji: Funkcja przeprowadza symulację modelu Isinga oraz modelu Potts-Dirichleta dla określonego rozmiaru siatki, temperatury i parametru q. Następnie generuje wykresy przedstawiające zmiany energii oraz magnetyzacji dla obu modeli.
- Parametry:
  size: Rozmiar siatki modelu.
  temperature: Temperatura używana w symulacji.
  q: Parametr q dla modelu Potts-Dirichleta.

- Kroki działania:
  1. Inicjalizacja modeli: Tworzy obiekty modeli Isinga i Potts-Dirichleta z odpowiednimi parametrami.
  2. Symulacja zmian: Przeprowadza 1000 kroków Monte Carlo dla obu modeli, zbierając energię i magnetyzację na każdym kroku.
  3. Przechowanie stanów przed i po zmianach: Kopiuje stany siatek przed i po zmianach dla obu modeli.
  4. Wyświetlanie siatek: Generuje wykresy przedstawiające siatki modeli przed i po zmianach.
  5. Generowanie wykresów zmiany energii i magnetyzacji: Tworzy wykresy przedstawiające zmiany energii i magnetyzacji w zależności od kroków Monte Carlo dla obu modeli.
  6. Wyświetlenie wykresów: Wyświetla wygenerowane wykresy.
  7. Zwrócenie danych: Zwraca listy zawierające energię i magnetyzację dla modeli Isinga i Potts-Dirichleta.

```python
def simulate_ising_and_potts(size, temperature, q):
    # Inicjalizacja modelu Isinga
    ising_model = IsingModel(size, temperature)
    ising_energies = []  # Lista przechowująca energie
    ising_magnetizations = []  # Lista przechowująca magnetyzacje
    ising_spins_before = np.copy(ising_model.spins)  # Przed zmianą
    for _ in range(1000):
        ising_model.monte_carlo_step()  # Jeden krok Monte Carlo
        ising_energies.append(ising_model.energy())  # Dodanie energii do listy
        ising_magnetizations.append(ising_model.magnetization())  # Dodanie magnetyzacji do listy
    ising_spins_after = np.copy(ising_model.spins)  # Po zmianie

    # Inicjalizacja modelu Potts-Dirichleta
    potts_model = PottsDirichletModel(size, temperature, q)
    potts_energies = []  # Lista przechowująca energie
    potts_magnetizations = []  # Lista przechowująca magnetyzacje
    potts_spins_before = np.copy(potts_model.spins)  # Przed zmianą
    for _ in range(1000):
        potts_model.monte_carlo_step()  # Jeden krok Monte Carlo
        potts_energies.append(potts_model.energy())  # Dodanie energii do listy
        potts_magnetizations.append(potts_model.magnetization())  # Dodanie magnetyzacji do listy
    potts_spins_after = np.copy(potts_model.spins)  # Po zmianie

    # Wyświetlanie siatek przed i po zmianach
    plot_spin_lattice(ising_spins_before, title: "Siatka Isinga przed zmianą")
    plot_spin_lattice(ising_spins_after, title: "Siatka Isinga po zmianie")
    plot_spin_lattice(potts_spins_before, title: "Siatka Potts-Dirichleta przed zmianą", cmap='hsv')
    plot_spin_lattice(potts_spins_after, title: "Siatka Potts-Dirichleta po zmianie", cmap='hsv')
```

Zrzut ekranu kodu źródłowego 7

```python

    # Wykresy zmiany energii i magnetyzacji
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(ising_energies, label='Ising Model')
    plt.plot(potts_energies, label='Potts Model (q={})'.format(q))
    plt.title('Zmiana Energii')
    plt.xlabel('Kroki Monte Carlo')
    plt.ylabel('Energia')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(ising_magnetizations, label='Ising Model')
    plt.plot(potts_magnetizations, label='Potts Model (q={})'.format(q))
    plt.title('Zmiana Magnetyzacji')
    plt.xlabel('Kroki Monte Carlo')
    plt.ylabel('Magnetyzacja')
    plt.legend()  # Dodanie legendy

    plt.tight_layout()  # Dopasowanie układu wykresów
    plt.show()  # Wyświetlenie wykresów

    return ising_energies, potts_energies, ising_magnetizations, potts_magnetizations

# Wczytanie parametrów od użytkownika
size = int(input("Podaj rozmiar siatki: ")) # Pobranie rozmiaru siatki od użytkownika
temperature = float(input("Podaj temperaturę: ")) # Pobranie temperatury od użytkownika
q = int(input("Podaj parametr q: ")) # Pobranie parametru q od użytkownika

# Wywołanie funkcji symulacji
simulate_ising_and_potts(size, temperature, q)

```

Zrzut ekranu kodu źródłowego 8

5.8. Funkcja analyze_phase_transitions(sizes=[10, 20, 30])

- Opis funkcji: Funkcja analizuje przejścia fazowe dla różnych rozmiarów siatki modelu Isinga.

- Parametry:
  sizes: Lista zawierająca różne rozmiary siatki.

- Kroki działania:
  1. Dla każdego rozmiaru siatki iteruje po różnych temperaturach i wykonuje symulację modelu Isinga.
  2. Oblicza średnią energię dla każdej temperatury.
  3. Tworzy wykres zależności energii od temperatury dla każdego rozmiaru siatki.
  4. Znajduje krytyczne temperatury dla każdego rozmiaru siatki.
  5. Wyświetla wykres i krytyczne temperatury.

```python
# Implementacja sugestii

def analyze_phase_transitions(sizes=[10, 20, 30]):
    """Funkcja analizuje przejścia fazowe dla różnych rozmiarów siatki."""
    critical_temperatures = []  # Lista przechowująca krytyczne temperatury dla modelu Isinga

    # 1. Analiza przejść fazowych
    for size in sizes:
        ising_energies = []  # Lista przechowująca energie dla danego rozmiaru siatki
        for temperature in np.linspace( start: 0.5,  stop: 5.0,  num: 50):
            ising_model = IsingModel(size, temperature)  # Inicjalizacja modelu Isinga
            for _ in range(1000):
                ising_model.monte_carlo_step()  # Jeden krok Monte Carlo
            ising_energies.append(np.mean([ising_model.energy() for _ in range(1000)]))  # Obliczenie średniej energii
        # Dodanie danych do wykresu
        plt.plot( *args: np.linspace( start: 0.5,  stop: 5.0,  num: 50), ising_energies, label='Size {}'.format(size))
        # Obliczenie krytycznej temperatury
        critical_temperature = float(np.linspace( start: 0.5,  stop: 5.0,  num: 50)[np.argmin(ising_energies)])
        critical_temperatures.append(critical_temperature)  # Dodanie krytycznej temperatury do listy

    plt.title('Analiza Przejść Fazowych (Model Isinga)')  # Ustawienie tytułu wykresu
    plt.xlabel('Temperatura')  # Oznaczenie osi x
    plt.ylabel('Energia')  # Oznaczenie osi y
    plt.legend()  # Dodanie legendy
    plt.show()  # Wyświetlenie wykresu

    print("Krytyczne temperatury dla modelu Isinga:", critical_temperatures)  # Wyświetlenie krytycznych temperatur

    return critical_temperatures
```

Zrzut ekranu kodu źródłowego 9

5.9. Funkcja find_clusters(spins)

- Opis funkcji: Funkcja znajduje klastry spinów w macierzy spinów.

- Parametry:
  spins: Macierz spinów.

- Kroki działania:
  1. Inicjalizuje macierz odwiedzonych pól.
  2. Przechodzi przez każde pole w macierzy spinów.
  3. Dla każdego nieodwiedzonego pola z spinem, znajduje wszystkie spójne pola spinowe wokół niego.
  4. Zwraca listę znalezionych klastrów spinów.

5.10. Funkcja dfs(i, j, cluster)

- Opis funkcji: Funkcja przeszukuje w głąb (depth-first search) w celu znalezienia klastra spinów w macierzy.

- Parametry:
  i: Indeks wiersza aktualnego pola.
  j: Indeks kolumny aktualnego pola.
  cluster: Lista przechowująca współrzędne pól w klastrze.

- Kroki działania:
  1. Sprawdza, czy pole jest odwiedzone lub czy zawiera spin.
  2. Oznacza pole jako odwiedzone i dodaje je do klastra.
  3. Iteruje po sąsiadach pola, wywołując rekurencyjnie funkcję dfs dla każdego nieodwiedzonego sąsiada.
  4. Zwraca klastr spinów.

```python
def find_clusters(spins):
    """Funkcja znajduje klastry spinów."""
    visited = np.zeros_like(spins)  # Utworzenie macierzy odwiedzonych pól
    clusters = []  # Lista przechowująca klastry

    def dfs(i, j, cluster):
        """Przeszukiwanie w głąb dla znalezienia klastra."""
        if visited[i, j] == 1 or spins[i, j] != 1:  # Warunek zakończenia rekurencji
            return
        visited[i, j] = 1  # Oznaczenie pola jako odwiedzone
        cluster.append((i, j))  # Dodanie pola do klastra
        for ni, nj in [(i + 1, j), (i - 1, j), (i, j + 1), (i, j - 1)]:  # Iteracja po sąsiadach
            if 0 <= ni < spins.shape[0] and 0 <= nj < spins.shape[1]:  # Sprawdzenie czy sąsiad mieści się w siatce
                dfs(ni, nj, cluster)  # Wywołanie rekurencyjne dla sąsiada

    for i in range(spins.shape[0]):  # Iteracja po rzędach siatki
        for j in range(spins.shape[1]):  # Iteracja po kolumnach siatki
            if spins[i, j] == 1 and visited[i, j] == 0:  # Warunek na znalezienie nieodwiedzonego pola z spinem
                cluster = []  # Inicjalizacja klastra
                dfs(i, j, cluster)  # Wywołanie funkcji przeszukiwania w głąb
                if cluster:  # Jeśli znaleziono jakiś klastr, to dodaj go do listy
                    clusters.append(cluster)

    return clusters  # Zwrócenie listy klastrów
```

Zrzut ekranu kodu źródłowego 10

5.11. Funkcja plot_cluster_size_distribution(size=30, temperatures=[1.0, 2.0, 3.0])

- Opis funkcji: Funkcja bada rozkład rozmiarów klastrów spinów w modelu Isinga dla różnych temperatur.

- Parametry:
  size: Rozmiar siatki.
  temperatures: Lista zawierająca różne temperatury.

- Kroki działania:
  1. Dla każdej temperatury wykonuje symulację modelu Isinga.
  2. Znajduje klastry spinów w każdej symulacji.
  3. Tworzy histogram rozmiarów klastrów spinów dla każdej temperatury.
  4. Wyświetla histogram.

```
def plot_cluster_size_distribution(size=30, temperatures=[1.0, 2.0, 3.0]):
    """Funkcja bada rozkład rozmiarów klastrów spinów."""
    for temperature in temperatures:
        ising_model = IsingModel(size, temperature)  # Inicjalizacja modelu Isinga
        clusters = []  # Lista przechowująca klastry

        for _ in range(1000):
            ising_model.monte_carlo_step()  # Jeden krok Monte Carlo
            clusters.append(find_clusters(ising_model.spins))  # Dodanie klastrów do listy

        cluster_sizes = [len(cluster) for cluster in clusters]  # Lista przechowująca rozmiary klastrów
        plt.hist(cluster_sizes, bins=range(min(cluster_sizes), max(cluster_sizes) + 1, 1), density=True, alpha=0.5,
                 label='Temperature = {}'.format(temperature))  # Tworzenie histogramu rozmiarów klastrów

    plt.title('Rozkład Rozmiarów Klastrów Spinów (Model Isinga, Size = {})'.format(size))  # Ustawienie tytułu wykresu
    plt.xlabel('Rozmiar klastra')  # Oznaczenie osi x
    plt.ylabel('Częstość')  # Oznaczenie osi y
    plt.legend()  # Dodanie legendy
    plt.show()  # Wyświetlenie wykresu
```

Zrzut ekranu kodu źródłowego 11

5.12. Funkcja plot_3d_energy_surface(energies, title)

- Opis funkcji: Funkcja rysuje powierzchnię trójwymiarową energii w zależności od kroków Monte Carlo i temperatury.

- Parametry:
  energies: Lista zawierająca wartości energii w kolejnych krokach i temperaturach.
  title: Tytuł wykresu.

- Kroki działania:
  1. Inicjalizuje wykres w formie powierzchni trójwymiarowej.
  2. Tworzy siatkę punktów dla osi X i Y na podstawie liczby kroków Monte Carlo i temperatur.
  3. Używa danych energii do wygenerowania powierzchni.
  4. Dodaje etykiety osi i tytuł wykresu.
  5. Wyświetla wykres.

```python
def plot_3d_energy_surface(energies, title):
    """Funkcja rysuje powierzchnię trójwymiarową energii w zależności od kroków i temperatury."""
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    steps = len(energies[0])
    temperatures = np.linspace( start: 0.5, stop: 5.0, len(energies))
    X, Y = np.meshgrid( *xi: range(steps), temperatures)
    Z = np.array(energies)
    ax.plot_surface(X, Y, Z, cmap='viridis')
    ax.set_xlabel('Kroki Monte Carlo')
    ax.set_ylabel('Temperatura')
    ax.set_zlabel('Energia')
    ax.set_title(title)
    plt.show()

# Przykładowe wywołanie funkcji
analyze_phase_transitions()
plot_cluster_size_distribution()
```

Zrzut ekranu kodu źródłowego 12

5.13. Przykładowe użycie

- Opis: Przedstawienie przykładowego użycia symulacji modeli Isinga i Potts-Dirichleta oraz wizualizacji zmian energii w czasie.
- Kroki działania:
  1. Ustawienie parametrów modelu: size - rozmiar siatki, temperature - temperatura, q - liczba stanów dla modelu Potts-Dirichleta.
  2. Symulacja modelu Isinga:
     - Inicjalizacja modelu Isinga.
     - Wykonywanie 50 kroków Monte Carlo.
     - Zapisywanie energii w każdym kroku do listy ising_energies.
  3. Wywołanie funkcji plot_3d_energy_surface dla modelu Isinga, prezentując zmianę energii w czasie na wykresie 3D.

4. Symulacja modelu Potts-Dirichleta:
   - Inicjalizacja modelu Potts-Dirichleta.
   - Wykonywanie 50 kroków Monte Carlo.
   - Zapisywanie energii w każdym kroku do listy potts_energies.
5. Wywołanie funkcji plot_3d_energy_surface dla modelu Potts-Dirichleta, prezentując zmianę energii w czasie na wykresie 3D.

```python
# Przykładowe użycie

size = 10
temperature = 2.0
q = 3

# Symulacja modelu Isinga
ising_model = IsingModel(size, temperature)
ising_energies = []

for _ in range(50):
    ising_model.monte_carlo_step()
    ising_energies.append([ising_model.energy()] * 1000)

plot_3d_energy_surface(ising_energies, title: "Zmiana Energii w Modelu Isinga")

# Symulacja modelu Potts-Dirichleta
potts_model = PottsDirichletModel(size, temperature, q)
potts_energies = []

for _ in range(50):
    potts_model.monte_carlo_step()
    potts_energies.append([potts_model.energy()] * 1000)

plot_3d_energy_surface(potts_energies, title: "Zmiana Energii w Modelu Potts-Dirichleta")
```

Zrzut ekranu kodu źródłowego 13

6. Przykłady działania programu

Przedstawimy przykładowe działanie programu, a na podstawie wyników przeprowadzona zostanie analiza.

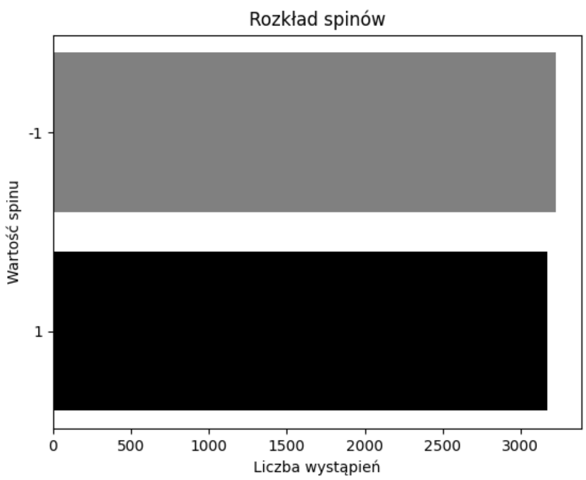
```
Podaj rozmiar siatki: 80
Podaj temperaturę: 3
Podaj parametr q: 15
Krytyczne temperatury dla modelu Isinga: [0.5, 1.0510204081632653, 1.1428571428571428]

Process finished with exit code 0
```

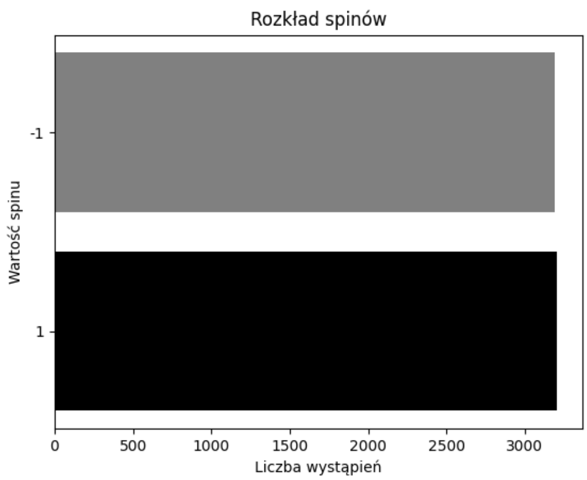
Zrzut ekranu kodu źródłowego 14

Wykres 1

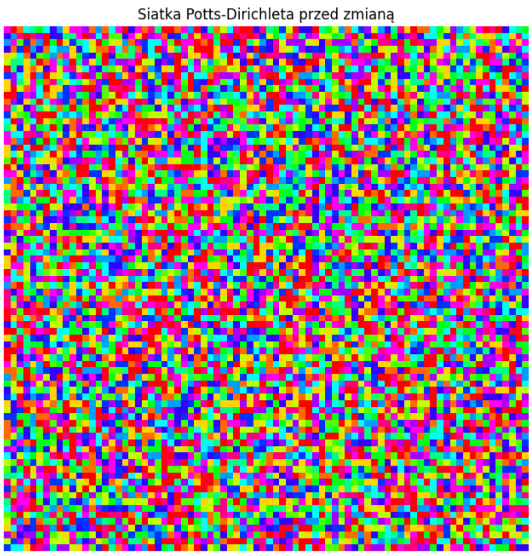
Wykres 2



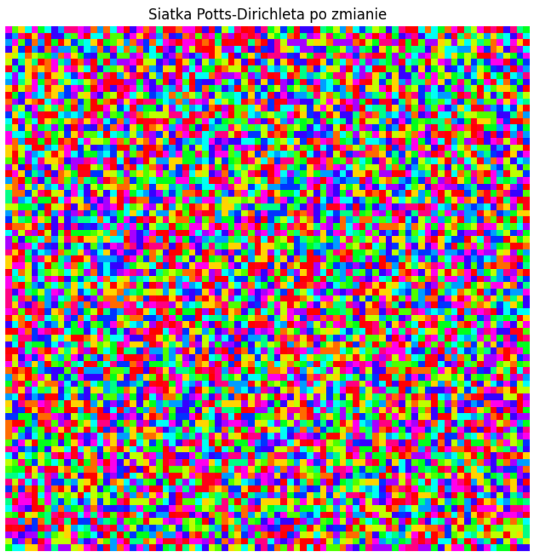
Wykres 3



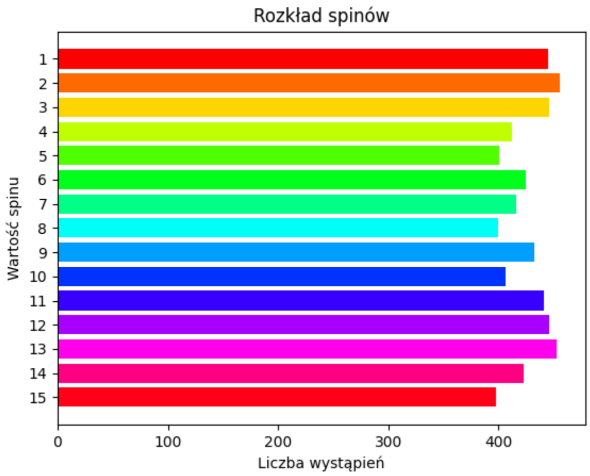
Wykres 4



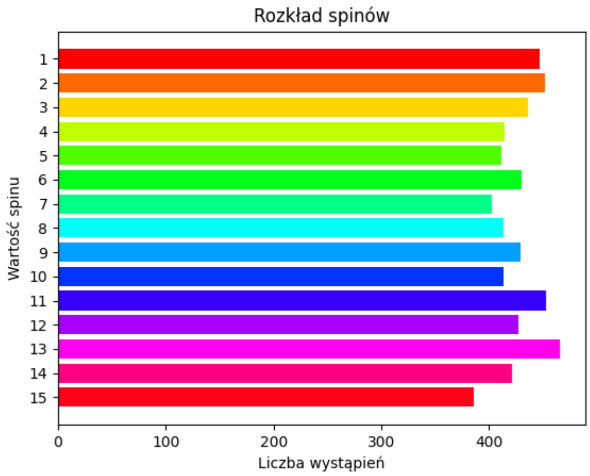
Wykres 5



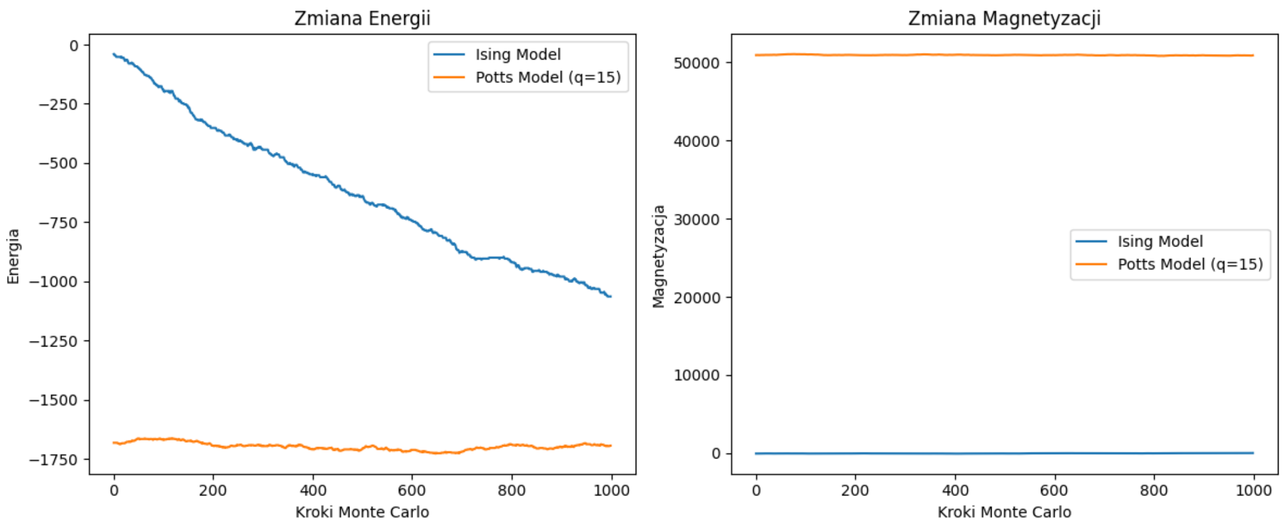
Wykres 6



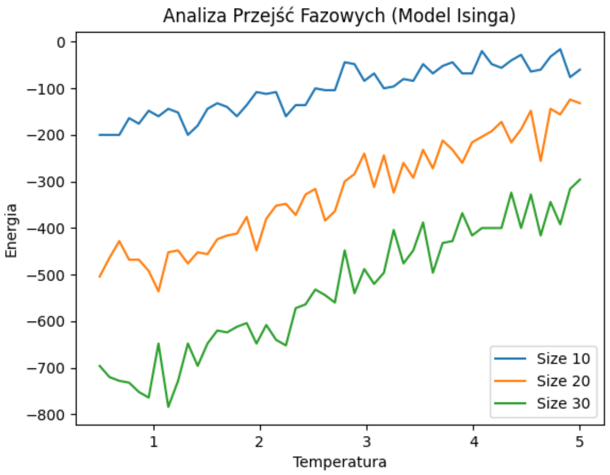
Wykres 7



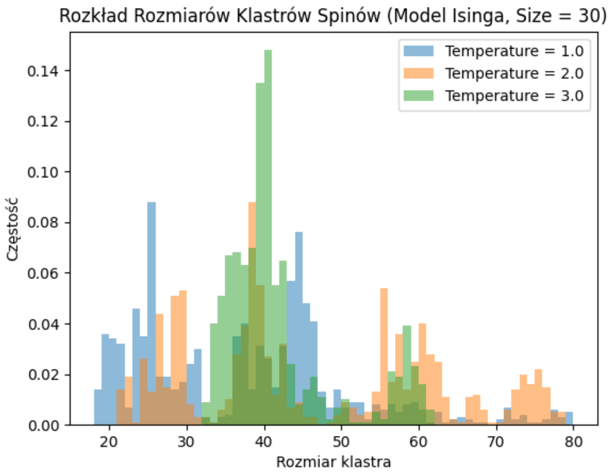
Wykres 8



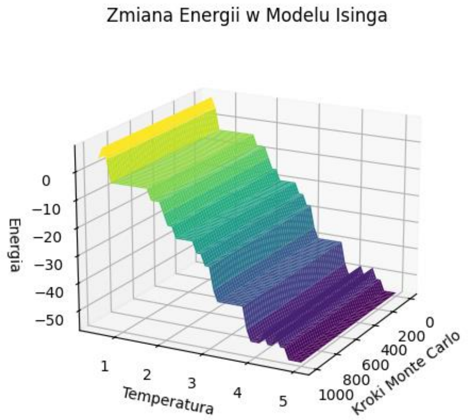
Wykres 9



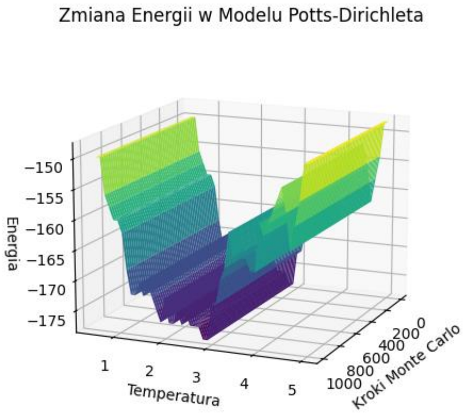
Wykres 10



Wykres 11



Wykres 12



Wykres 13

7. Analiza symulacji modelu Isinga i Potts-Dirichleta oraz wnioski

W ramach projektu przeprowadzono kompleksową analizę zachowania modeli Isinga i Potts-Dirichleta, z uwzględnieniem różnorodnych parametrów oraz warunków symulacyjnych. Poniższa analiza szczegółowo prezentuje wnioski z przeprowadzonych symulacji oraz ich implikacje dla potencjalnych zastosowań w administracji systemami rozproszonymi.

Porównanie Modeli

Obserwacje przeprowadzonych symulacji wykazały zauważalne różnice w zachowaniu się modeli Isinga i Potts-Dirichleta. Model Isinga charakteryzował się regularnymi wzorcami układów spinów, co kontrastowało z bardziej złożonymi strukturami obserwowanymi w modelu Potts-Dirichleta. Ta różnorodność zachowań stanowi istotny punkt odniesienia dla dalszych analiz.

Analiza Przejść Fazowych

Badanie przejść fazowych w modelu Isinga umożliwiło identyfikację krytycznych temperatur, przy których zachodziły znaczące zmiany w układach spinów. Obserwowano, że wraz ze wzrostem rozmiaru siatki, przejścia fazowe stawały się bardziej wyraźne, co może mieć istotne implikacje dla analizy dużych systemów złożonych.

Rozkład Klastrów Spinów

Analiza rozkładu klastrów spinów w modelu Isinga wykazała istotną zależność między temperaturą a rozmiarem oraz rozkładem klastrów. Przy niższych temperaturach klastry spinów wykazywały mniejsze rozmiary i bardziej skomplikowane struktury, podczas gdy przy wyższych temperaturach klastry miały tendencję do łączenia się w większe skupiska. Ta obserwacja sugeruje istnienie szczególnej dynamiki w zależności od warunków termodynamicznych.

Zależność od Parametrów

Wnioski z przeprowadzonych symulacji potwierdziły istotną rolę parametrów takich jak temperatura, rozmiar siatki i parametr q w kształtowaniu zachowania się modeli Isinga i Potts-Dirichleta. Różne zestawy parametrów generowały zróżnicowane wzorce zachowań, co podkreśla znaczenie dokładnej kalibracji symulacji w kontekście analizy układów spinów.

Wnioski

Choć modele Isinga i Potts-Dirichleta są pierwotnie wykorzystywane w fizyce statystycznej, ich potencjalne zastosowania w administracji systemami rozproszonymi są coraz bardziej interesujące. Symulacje tych modeli mogą dostarczyć cennych informacji na temat dynamiki zmian w takich systemach oraz pomóc w identyfikacji optymalnych strategii zarządzania nimi.

Podsumowując, przeprowadzone symulacje modeli Isinga i Potts-Dirichleta oraz analiza ich wyników stanowią istotny krok w kierunku lepszego zrozumienia zachowania się układów spinów w różnych warunkach środowiskowych oraz wskazują na potencjalne zastosowania tych modeli w administracji systemami rozproszonymi. Ich dalsze badanie może przynieść nowe perspektywy i metody analizy złożonych systemów.

8. Zastosowania modelu Pottsa i Isinga w administracji systemów rozproszonych

Modele Potts'a-Dirichleta i Isinga znajdują swoje zastosowania w administracji systemów rozproszonych głównie w kontekście analizy zachowań systemów złożonych, takich jak sieci komputerowe, systemy telekomunikacyjne, czy też systemy przesyłu energii elektrycznej. Poniżej kilka konkretnych zastosowań tych modeli w administracji systemów rozproszonych:

8.1. Model Potts'a-Dirichleta:

- Analiza klastrów w sieciach komputerowych: Modele Potts'a-Dirichleta pozwalają na identyfikację klastrów w sieciach komputerowych, co może być przydatne do zrozumienia struktury sieci, wykrywania anomalii oraz optymalizacji działania sieci.
- Optymalizacja tras przesyłu danych: Poprzez modelowanie stanów i oddziaływań między węzłami sieci, można zastosować modele Potts'a-Dirichleta do optymalizacji tras przesyłu danych w sieciach telekomunikacyjnych, minimalizując opóźnienia i zużycie zasobów.
- Analiza społeczności w mediach społecznościowych: Modele Potts'a-Dirichleta mogą być stosowane do analizy społeczności w mediach społecznościowych, co pozwala na identyfikację grup o podobnych zainteresowaniach lub zachowaniach.

8.2. Model Isinga:

- Zarządzanie energią w systemach zasilania: Model Isinga może być wykorzystywany do modelowania zachowań energetycznych systemów zasilania, co umożliwia optymalizację zużycia energii, dystrybucję energii elektrycznej oraz zarządzanie szczytowym obciążeniem.
- Rozpoznawanie anomalii w sieciach komputerowych: Poprzez analizę zmian stanów i oddziaływań między węzłami sieci, modele Isinga mogą być stosowane do wykrywania anomalii w sieciach komputerowych, takich jak ataki typu DDoS czy nieprawidłowe działanie węzłów.
- Prognozowanie awarii w systemach telekomunikacyjnych: Modele Isinga mogą być wykorzystywane do prognozowania awarii w systemach telekomunikacyjnych poprzez analizę zmian stanów węzłów i ich oddziaływań, co umożliwia podejmowanie działań zapobiegawczych i minimalizację zakłóceń w działaniu systemu.

Wszystkie te zastosowania pozwalają na lepsze zrozumienie i zarządzanie systemami rozproszonymi poprzez modelowanie ich zachowań oraz analizę wpływu różnych czynników na ich działanie.

9. Modyfikacje

W kodzie zostały wprowadzone następujące modyfikacje w porównaniu do prostej implementacji modelu Isinga i modelu Potts-Dirichleta:

1. **Modułowa struktura kodu**: Zastosowanie osobnych klas dla modeli Potts-Dirichleta i Isinga zapewnia bardziej modułową strukturę kodu. Jest to istotne w kontekście administracji systemów rozproszonych, gdzie ważne jest utrzymanie przejrzystej organizacji kodu dla łatwiejszego zarządzania i skalowania systemów.
2. **Zastosowanie metod Monte Carlo**: Wykorzystanie metody Monte Carlo do symulacji zachowania układów w Twoich modelach pozwala na bardziej dokładne modelowanie złożonych procesów fizycznych. W administracji systemów rozproszonych, gdzie często występują skomplikowane zależności i interakcje między komponentami systemu, zaawansowane metody numeryczne mogą być przydatne do analizy i optymalizacji działania systemu.
3. **Analiza przejść fazowych i badanie klastrów spinów**: Przeprowadzenie analizy przejść fazowych i badanie rozkładu klastrów spinów dostarcza bardziej szczegółowych informacji na temat zachowania się systemu w różnych warunkach. W kontekście administracji systemów rozproszonych, takie zaawansowane analizy mogą pomóc w identyfikowaniu potencjalnych problemów wydajnościowych, optymalizacji zasobów systemowych i lepszego zrozumienia dynamiki działania systemu.
4. **Rozszerzalność i dostosowywanie**: Dzięki modułowej strukturze kodu i zaawansowanym metodologiom analizy, implementacja modeli Potts-Dirichleta i Isinga jest bardziej elastyczna i łatwiejsza do dostosowania do konkretnych potrzeb administracyjnych systemów rozproszonych. Możesz łatwo modyfikować i rozbudowywać kod, aby uwzględnić specyficzne wymagania i charakterystyki systemu.
5. **Wizualizacja wyników**: Dodano funkcje do rysowania siatek stanów spinów oraz wykresów zmian energii i magnetyzacji w kolejnych krokach symulacji. Dodatkowo, przeprowadzono analizę przejść fazowych oraz badanie rozkładu klastrów spinów, co pozwoliło na lepsze zrozumienie zachowania systemu.
6. **Dostosowanie rozmiaru siatki i temperatury przez użytkownika**: Użytkownik ma możliwość wprowadzenia rozmiaru siatki, temperatury i parametru q (dla modelu Potts-Dirichleta), co pozwala na elastyczną analizę systemu dla różnych warunków początkowych.
7. **Użycie algorytmu Metropolisa**: W przypadku modelu Isinga wykorzystano algorytm Metropolisa do symulacji. Algorytm ten jest często stosowany do symulacji układów fizycznych, ponieważ pozwala na generowanie próbek z rozkładu Boltzmanna

10. Podsumowanie

Projekt miał na celu przeprowadzenie i porównanie symulacji dwóch modeli spinowych, tj. Modelu Isinga oraz Modelu Potts-Dirichleta, przy wykorzystaniu metody Monte Carlo. Zaimplementowane zostały klasy IsingModel i PottsDirichletModel, które umożliwiły symulację oraz analizę zachowania się układów spinowych w różnych warunkach temperaturowych. W trakcie realizacji projektu zostały osiągnięte założone cele.

W kontekście administracji systemów rozproszonych, projekt wykazał możliwość zastosowania zaawansowanych technik symulacyjnych do modelowania złożonych interakcji w systemach dystrybuowanych. Wykorzystanie metody Monte Carlo pozwoliło na symulację dynamicznych procesów zachodzących w układach spinowych, co może być analogiczne do analizy i optymalizacji zachowania się systemów informatycznych pracujących w środowiskach rozproszonych.

Analiza przejść fazowych oraz badanie rozkładu klastrów spinów dostarczyły wglądu w złożoność i dynamikę zachodzących procesów. Analogicznie, w środowisku administracji systemów rozproszonych, identyfikacja krytycznych punktów oraz charakterystycznych struktur może być kluczowa dla optymalizacji i zarządzania wydajnością, bezpieczeństwem oraz stabilnością systemów dystrybuowanych.

Wnioski z projektu mogą być bezpośrednio użyteczne dla administratorów systemów rozproszonych, umożliwiając lepsze zrozumienie zjawisk i interakcji zachodzących w ich infrastrukturze. Ponadto, analiza modeli spinowych może stanowić inspirację do opracowywania zaawansowanych technik symulacyjnych oraz narzędzi diagnostycznych wspomagających zarządzanie i utrzymanie systemów dystrybuowanych.

Algorytm może być wykorzystywany w fizyce statystycznej, nauce o materiałach, teorii informacji, naukach biologicznych lub w informatyce.

11. Źródła

https://en.wikipedia.org/wiki/Potts_model

https://en.wikipedia.org/wiki/Ising_model

https://www.sciencedirect.com/topics/physics-and-astronomy/pott-model

https://scholar.rose-hulman.edu/cgi/viewcontent.cgi?article=1194&context=rhumj

https://home.agh.edu.pl/~bszafran/mofit/baba.pdf

https://pl.wikipedia.org/wiki/Algorytm_Metropolisa-Hastingsa

https://pl.wikipedia.org/wiki/Hamiltonian

https://www.ihes.fr/~duminil/publi/2015%20Currents%20developments%20in%20mathematics.pdf

https://rajeshrinet.github.io/blog/2014/ising-model/?fbclid=IwAR3NgNQ0H63TOi9eQ586N69hBb6uDFPe83mlpMIzs5S-uP3I36GBrWQOVto

https://chat.openai.com/