Sztuczna inteligencja i inżynieria wiedzy

Z00-32m

poniedziałek 1515 – 1655

Karolina Lenczewska 227993

**Sprawozdanie 2.**

**PROBLEMY SPEŁNIENIA OGRANICZEŃ**

# Cel sprawozdania

Zapoznanie się oraz przebadanie podstawowych algorytmów stosowanych do rozwiązywania problemów spełniania ograniczeń – CSP.

# Opis problemów

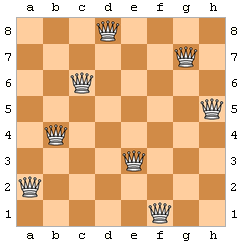
Do wykonania badań związanych z podstawowymi algorytmami stosowanymi do rozwiązywania problemów spełniania ograniczeń, decydowałam się na wybór dwóch następujących problemów z trzech zaproponowanych na liście:

* N-hetmanów
* Kwadrat łaciński

Każdy z wybranych problemów spełnia warunki bycia problemem spełnienia ograniczeń, poprzez posiadanie trzech następujących składowych: zmienne, dziedziny oraz ograniczenia.

## Problem n-hetmanów

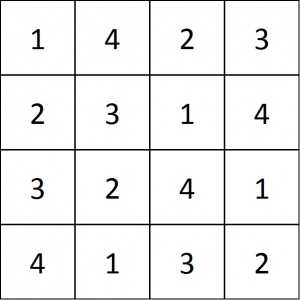
Zadaniem w problemie N-hetmanów jest odpowiednie umiejscowienie N hetmanów na planszy o wymiarach N x N, w taki sposób aby wzajemnie się nie atakowały – figury nie mogą znajdować się w tym samym wierszu, kolumnie oraz na przekątnej. Przykład rozwiązania problemu N-hetmanów:



Składowe w problemie N-hetmanów:

* **Zmienne:** N kolumn umiejscowionych w liście obiektów reprezentujących pola na planszy. Każda zmienna posiada własną dziedzinę.
* **Dziedzina:** zbiór (0, N) wierszy. Liczba ‘-1’ oznacza kolumnę, na której nie został jeszcze umiejscowiony hetman. Pozostałe liczby [0,N) reprezentują wiersz, w którym został umiejscowiony hetman.
* **Ograniczenia:** w odpowiednim wierszu, kolumnie bądź skosie, gdzie został już poprzednio umiejscowiony hetman, nie jesteśmy w stanie postawić nowego hetmana

## problem kwadratu łacińskiego

Zadaniem w problemie kwadratu łacińskiego jest odpowiednie umiejscowienie cyfr należących do przedziału [0,N] na planszy o wymiarach N x N, w taki sposób aby w każdym wierszu oraz kolumnie nie występowała więcej niż raz ta sama liczba. Przykład rozwiązania problemu kwadratu łacińskiego:

Składowe w problemie N-hetmanów:

* **Zmienne:** pola na planszy N x N, z której każde posiada własną dziedzinę.
* **Dziedzina:** zbiór liczb zawierający się od 1 do N włącznie.
* **Ograniczenia:** W każdej kolumnie ora wierszu może nie może wystąpić więcej razy niż raz ta sama liczba.

# Sposoby i opisy rozwiązań problemu

## Algorytm przeszukiwania przyrostowego z powracaniem (eng. backtracking)

Algorytm przeszukiwania przyrostowego z powracaniem, systematycznie szuka rozwiązania pośród wszystkich dostępnych. Algorytm w implementacji startuje z pustą listą (dla hetmanów), bądź planszą (dla kwadratu łacińskiego), który wypełnia częściowymi rozwiązaniami. Kiedy dochodzi do napotkania z niepoprawnym rozwiązaniem, nawraca do punktu, z którego jest w stanie podjąć inną drogę w celu rozwiązania problemu.

Zaimplementowane metody dla danego algorytmu:

* *isHetmanPlacementOk() / isBoardOk()* – metody zaimplementowane odpowiednio dla potrzeb sprawdzenia spełnienia ograniczeń w problemach N-hetmanów oraz kwadratu łacińskiego
* *isVariableOk()* – metoda wykorzystana w przypadku problemu kwadratu łacińskiego, sprawdza czy wybrana zmienna istnieje na planszy, jeżeli nie (wtedy zarówno jej wiersz jak i kolumna równa się -2) algorytm zostaje zakończony.
* *initColumns() / initBoard()* – tworzy pustą planszę, która zostanie później wykorzystana do zapisywania częściowych zmiennych.
* *findFreeSpot()* - metoda wykorzystana w przypadku problemu kwadratu łacińskiego, przeszukuje całą planszę zaczynając z lewego górnego rogu w poszukiwaniu jeszcze nie użytych zmiennych.
* *solve\_BT\_4\_ALL()* – inicjalizuje działanie algorytmu, liczy czas jego działania oraz wypisuje na konsoli wyniki algorytmu.
* *solve\_Backtracking\_4\_ALL()* – metoda rekurencyjnie wywołująca algorytm.
* *reset()* – ustawia wartości zmiennych *solutions* oraz *nodes* spowrotem na 0 po wykonaniu algorytmu.
* *printColumns() / printBoard()* – wyświetla plansze wynikową

## algorytm sprawdzania wprzód (eng. forward checking)

Algorytm sprawdzania wprzód polega na porównaniu nowej zaproponowanej zmiennej z wartościami zmiennych jeszcze nie rozpatrywanych. Następnie eliminuje zmienne nie spełniające ograniczeń ze wszystkich dziedzin, z którymi łączy się rozpatrywana zmienna. Algorytm kończy się w chwili, kiedy którakolwiek dziedzina zmiennej stanie się pusta.

Zaimplementowane metody dla danego algorytmu:

* *isVariableOk()* – metoda wykorzystana w przypadku problemu kwadratu łacińskiego, sprawdza czy wybrana zmienna istnieje na planszy, jeżeli nie (wtedy zarówno jej wiersz jak i kolumna równa się -2) algorytm zostaje zakończony.
* *initColumns() / initBoard()* – tworzy pustą planszę, która zostanie później wykorzystana do zapisywania częściowych zmiennych.
* *findFreeSpot()* - metoda wykorzystana w przypadku problemu kwadratu łacińskiego, przeszukuje całą planszę zaczynając z lewego górnego rogu w poszukiwaniu jeszcze nie użytych zmiennych.
* *solve\_FC\_4\_ALL()* – inicjalizuje działanie algorytmu, liczy czas jego działania oraz wypisuje na konsoli wyniki algorytmu.
* *solve\_ForwardChecking\_4\_ALL()* – metoda rekurencyjnie wywołująca algorytm.
* *isDwo()* – metoda zwraca true, jeżeli którakolwiek dziedzina ze zmiennych jest pusta.
* *isMyDwo()* – metoda zwraca true, jeżeli dziedzina danej zmiennej jest pusta.
* *reset()* – ustawia wartości zmiennych *solutions* oraz *nodes* spowrotem na 0 po wykonaniu algorytmu.
* *printColumns() / printBoard()* – wyświetla plansze wynikową

# propozycja heurystyk

1. Heurystyki dla problemu N-Hetmanów
   1. Rozpoczęcie działania algorytmu z prawego końca planszy, poprzez pobieranie zmiennych z końca tablicy kolumn oraz rozpoczęcie przypisywania im numeru wiersza od końca (zaczyna wiersz N-1). – wybór zmiennej oraz wartości
   2. Zmienne wybierane od środka tablicy kolumn do końca, a następnie wybranie zmiennych z początku do końca. Jako że zaczniemy od zmiennych, które występują w dużej liczbie ograniczeń, istnieje możliwość przyspieszenia działania algorytmu *forward checking*. – wybór zmiennej
      1. Jeżeli tablica składa się z kolumn [0, 1, 2, 3, 4], to algorytm będzie pobierał zmienne w kolejności [2 , 3 , 4, 0, 1]
2. Heurystyki dla problemu kwadratu łacińskiego
   1. Zaczynamy algorytm od wyboru zmiennych ze środka planszy – zmienne wybierane są w sposób spiralny. Jako, że wybieramy na początku zmienne występujące w największej liczbie ograniczeń, to heurystyka powinna przyspieszyć działanie algorytmu *forward checking*. – wybór zmiennej
   2. Zaczynamy algorytm od wyboru zmiennych z lewego górnego rogu planszy, natomiast wybieramy je w sposób spiralny. Dodatkowo liczby będące dziedziną przypisujemy polom w planszy zaczynając najpierw od parzystych. – wybór zmiennej i wartości

# Opis modelu

## architektura:

### problem n-hetmanów

### problem kwadratu łacińskiego

## proces wykonywania się algorytmóww kodzie:

### problem n-hetmanów

#### backtracking

#### forwardchecing

### problem kwadratu łacińskiego

#### backtracking

#### forwardchecing

# Plan badań

## Plan:

## Dane wykorzystane do badań:

# Wyniki badań

# Wnioski