Sztuczna inteligencja i inżynieria wiedzy laboratorium

Sprawozdanie 2

*Problem spełniania ograniczeń*

Arkadiusz Marcinowski

228160

W8, Informatyka

1. **Wstęp**

Celem zadanie było zapoznanie się z podstawowymi algorytmami stosowanymi do rozwiązywania problemów spełniania ograniczeń (ang. Constraint Satisfaction Problem, CSP), poprzez własnoręczną  
implementację i zbadanie ich właściwości.

Problem N-hetmanów oraz kwadratu łacińskiego zdefiniowano jako problem CSP i rozwiązano wykorzystując algorytmy sprawdzenia w przód oraz przeszukiwania przyrostowego z powracaniem dla różnych wartości N, gdzie N definiuje wielkość wybranego problemu.

1. **Podejście do problemu**

**2.1. Przeszukiwanie przyrostowe**

* przeszukiwanie w głąb, każdy krok to ustalenie wartości jednej zmiennej
* kolejność przypisywania zmiennych jest ustalona
* jeśli ustalenie kolejnej zmiennej jest niewykonalne bez łamania więzów następuje powrót, tzn. cofnięcie niektórych przypisań

**2.2. Przeszukiwanie w przód**

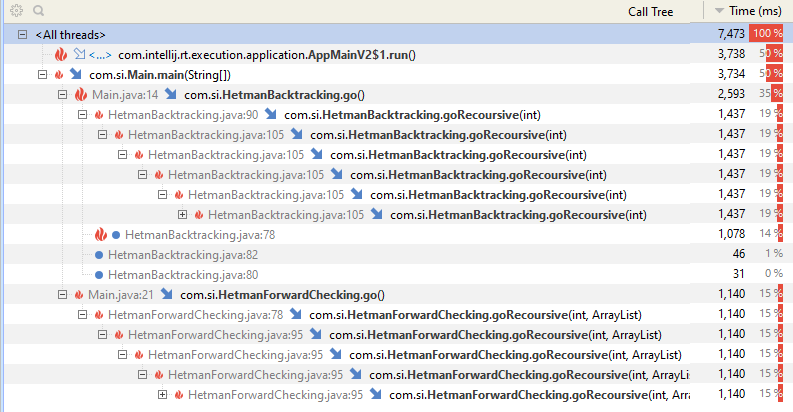
Nadając wartość pewnej zmiennej usuwamy z dziedziny każdej nieokreślonej jeszcze zmiennej te wartości, które są sprzeczne z wartościami zmiennych już określonych. Jeśli w wyniku usunięcia któraś z dziedzin stanie się pusta, wykonujemy nawrót.

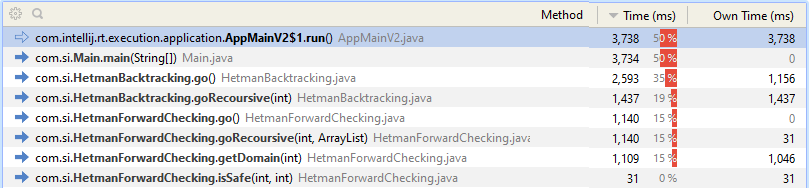
1. **Test działania**

Przeprowadzono test działania programu dla problemu N-hetmanów, z użyciem oprogramowania *YourKit Java Profiler.*

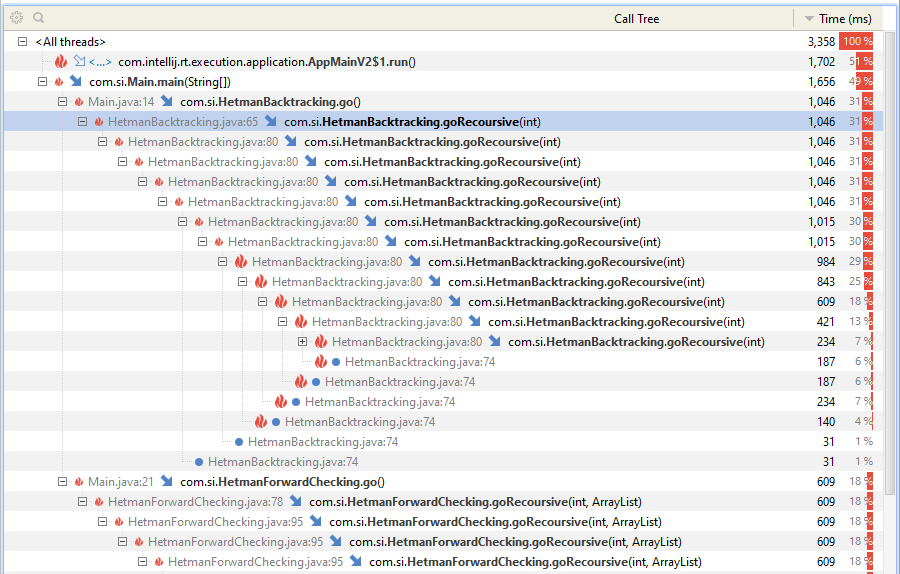
Wyniki pokazały, że najbardziej kosztowną metodą jest goRecoursive(). Poniższy screen przedstawia wyniki przed i po optymalizacji.

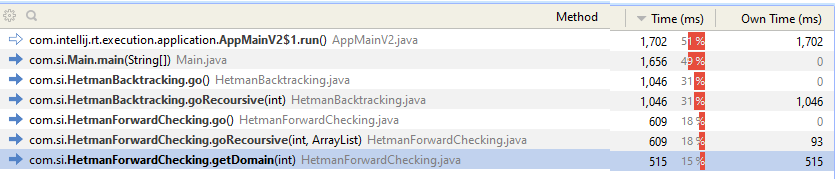
PRZED:





PO OPTYMALIZACJI:





1. **Badania**
   1. **N-Hetmanów**

Zależność czasu (nano s) pracy algorytmu od wielkości problemu, dla N-Hetmanów.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| BT | 8033 | 3570 | 5355 | 15172 | 57120 | 153062 | 413224 | 2845712 | 2173666 | 10025369 | 32850867 | 149391208 |
| FC | 16511 | 7139 | 13834 | 70953 | 164665 | 262393 | 1100889 | 2072368 | 3302223 | 1631878 | 32604539 | 159431304 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | 13 | 14 | 15 | 16 |
| BT | 916615758 | 798069985 | 31401111600 | 2,26001E+11 |
| FC | 798069985 | 4689280495 | 31234051270 | 2,01241E+11 |

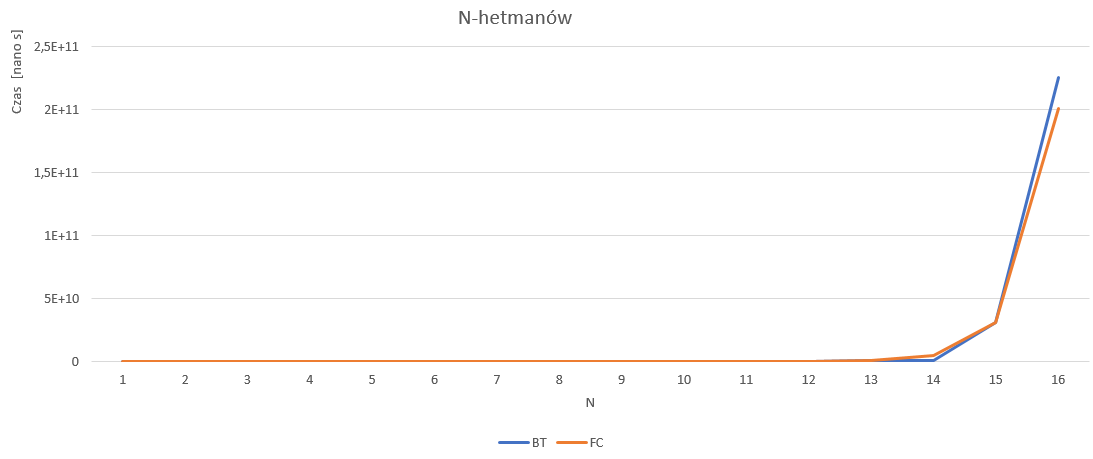
Zauważono, że do N mniejszego od 10, **przeszukiwanie przyrostowe** jest szybszym sposobem na znalezienie wszystkich rozwiązań. Dla N większego od 10, to **sprawdzanie w przód** notuje lepsze czasy.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| BT | 2 | 3 | 6 | 17 | 54 | 153 | 552 | 2057 | 8394 | 35539 | 166926 | 856189 | 4674890 | 27358553 | 171129072 | 1141190303 |
| FC | 2 | 1 | 3 | 13 | 50 | 107 | 398 | 1413 | 5810 | 23489 | 108530 | 555659 | 3031950 | 17693457 | 110889484 | 740447439 |

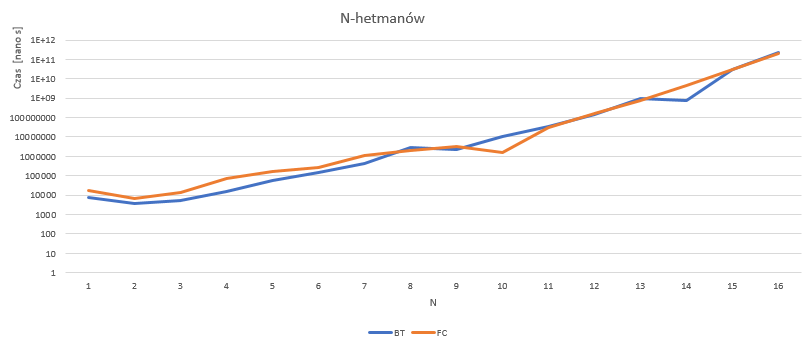
Zależność liczby odwiedzonych węzłów przez algorytm, od wielkości problemu, dla N-Hetmanów.

Zauważono, że **algorytm** **sprawdzania w przód** odwiedza dużo więcej węzłów, niż **algorytm przyrostowy**. Różnica jest niemal dwukrotna.

Analizując dwa powyższe zestawienia stwierdzono również, że **algorytm sprawdzania wprzód** spędza więcej czasu w każdym węźle, niż ma to miejsce w przypadku **algorytmu przyrostowego**.



W skali logarytmicznej:



* 1. **Kwadrat Łaciński**

Zależność czasu (ms) pracy algorytmu od wielkości problemu, dla Kwadratu Łacińskiego.

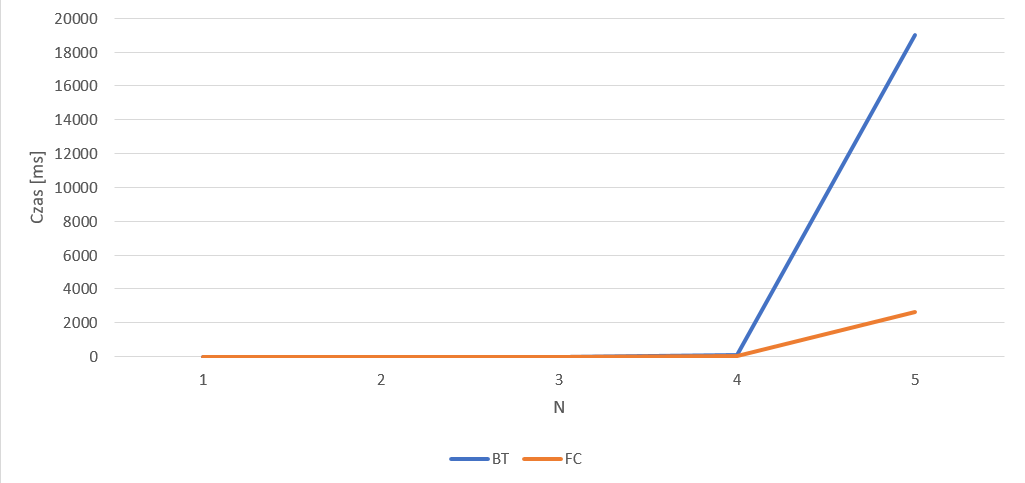
|  |  |  |  |
| --- | --- | --- | --- |
| N | 1 – 3 | 4 | 5 |
| BT | 0 | 85 | 19019 |
| FC | 0 | 16 | 2645 |

Zauważono, że **przeszukiwanie przyrostowe** jest szybszym sposobem na znalezienie wszystkich rozwiązań. **Sprawdzanie w przód** notuje wielokrotnie gorsze czasy.

Zależność liczby odwiedzonych węzłów przez algorytm, od wielkości problemu, dla Kwadratu Łacińskiego.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | 1 | 2 | 3 | 4 | 5 |
| BT | 2 | 15 | 247 | 20421 | 10764431 |
| FC | 2 | 9 | 94 | 5585 | 2178805 |

Zauważono, że **algorytm** **sprawdzania w przód** odwiedza dużo więcej węzłów, niż **algorytm przyrostowy**. Różnica jest nawet kilkukrotna dla badanych danych.

****

1. **Heurystyka**

**5.1. Punkt startu**

Pierwsze podejście to miana punktu startowego algorytmu BT dla N-hetmanów z lewego górnego rogu, na punkt [N/2][N/2], dla znalezienia pierwszego rozwiązania.

Punkt startowy- lewy górny róg

**============================== N QUEENS ==============================**

**BACKTRACKING**

**n = 13 Wywołań: 112 Rozwiązań: 1 Mediana z 10 uruchomień: 157079.0ns**

Punkt startowy- [N/2][N/2]

**============================== N QUEENS ==============================**

**BACKTRACKING**

**n = 10 Wywołań: 54 Rozwiązań: 0 Mediana z 3 uruchomień: 42393.0ns**

Po zmianie algorytm szybciej odnajduje pierwsze rozwiązanie, sprawdza mniej węzłów.

1. **Podsumowanie**

Algorytm sprawdzania wprzód i sprawdzania przyrostowego są bardzo ciekawymi sposobami rozwiązania przedstawionego zadania. Algorytmy znacznie różnią się od siebie.

Dla N-królowych, algorytm przyrostowy był szybszy jedynie do pewnego momentu. Po przekroczeniu N=10, jego czas rósł bardzo szybko, a algorytm sprawdzania wprzód zachowywał wysoką efektywność, przy nie aż tak rosnącym czasie. (Wykres został przedstawiony w treści zadania.)

Na działanie algorytmów wpływa ich implementacja oraz to jak bardzo panujemy nad kodem. Analizując dane z *profilera*, można było znaleźć miejsce gdzie złożoność działań była wysoka i usprawnić działanie algorytmu zyskując kolejne części sekundy. Poznanie narzędzia jakim jest *profiler* może pozwolić na pisanie bardziej efektywnego kodu w przyszłości.

Dobór odpowiednich heurystyk dla algorytmów może pomóc w optymalizacji działania programu przez ograniczenie węzłów jakie odwiedza algorytm.