

## Problem N-hetmanów.

Problem polega na ustawieniu na tablicy w ten sposób, aby każdy z hetmanów nie mógł zaatakować innego. Hetman jest w stanie zaatakować innego hetmana, gdy figura znajduje się na przekątnej, bądź na prostej, na której znajduje się inny hetman.

Problem jest problemem CSP (spełniania ograniczeń). W podanym przypadku zmiennymi są położenia królowej na planszy (mogą być prezentowane przez np. numer wiersza w kolumnie, gdzie należy umieścić królową), dziedziną (hetman/wolne pole), oraz więzy, tj. spełnienie poniższych założeń:

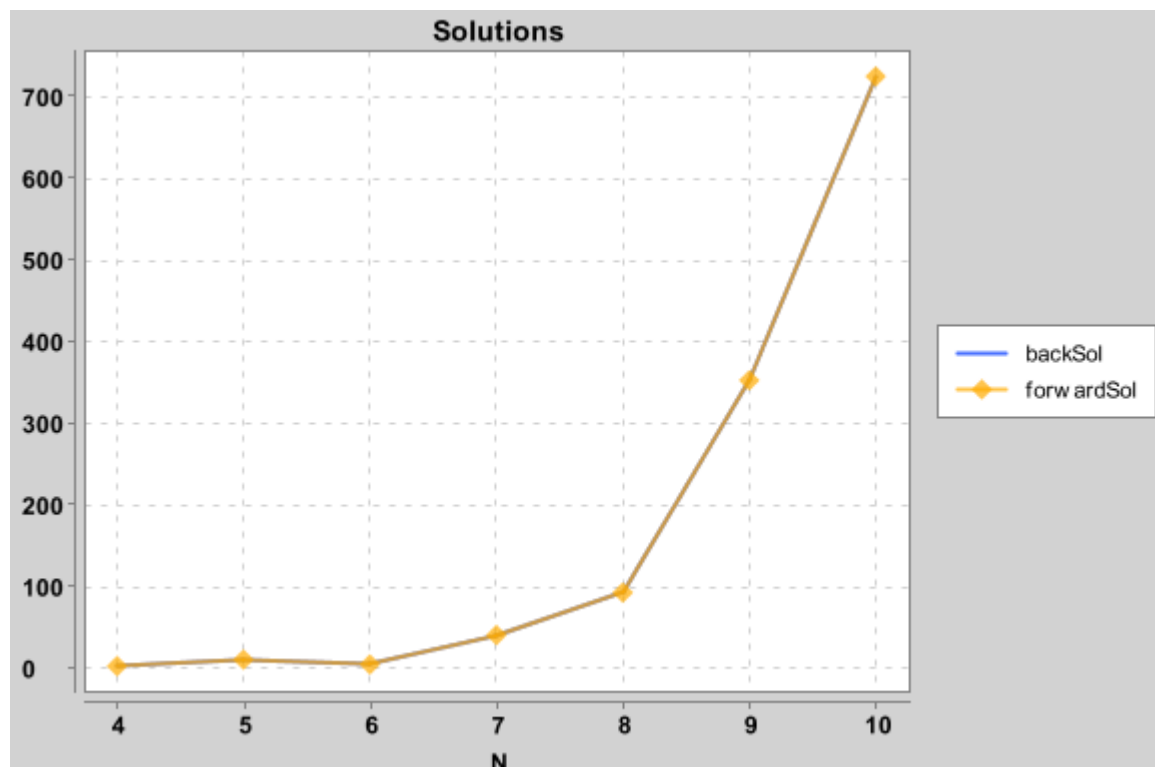
Niech  $(i,j)$  oraz  $(m,n)$  oznaczają współrzędne dwóch hetmanów. Dwa hetmany stoją na jednej linii wtedy i tylko wtedy  $i = m$  lub  $j = n$ . Dwa hetmany stoją na przekątnej wtedy i tylko wtedy, gdy  $i = m \pm x$  oraz  $j = n \pm x$ . Na jednej linii, przekątnej musi być dokładnie jeden hetman.

Problem można rozwiązać za pomocą wielu algorytmów, między innymi forward checking oraz backtracking.

Backtracking polega na tym, że algorytm szuka wszystkich rozwiązań, jednak podczas szukania kandydata, gdy stwierdzi, że ten nie może być rozwiązaniem nawraca do momentu, gdzie może podjąć inną budowę rozwiązania.

Forward checking różni się od niego, że zanim algorytm podejmie próbę budowania rozwiązania sprawdzi, czy jest ono możliwe.

Zarówno liczba rozwiązań jak i kombinacji nie jest liniowa dla zadanego problemu (dla przykładu – tablica 8x8 posiada 4 426 165 368 kombinacji, ale jedynie 92 rozwiązania). Dlatego oba algorytmy starają nakładać ograniczenia i sprawdzać tylko te potencjalne rozwiązania, które mogą przynieść pozytywny rezultat.



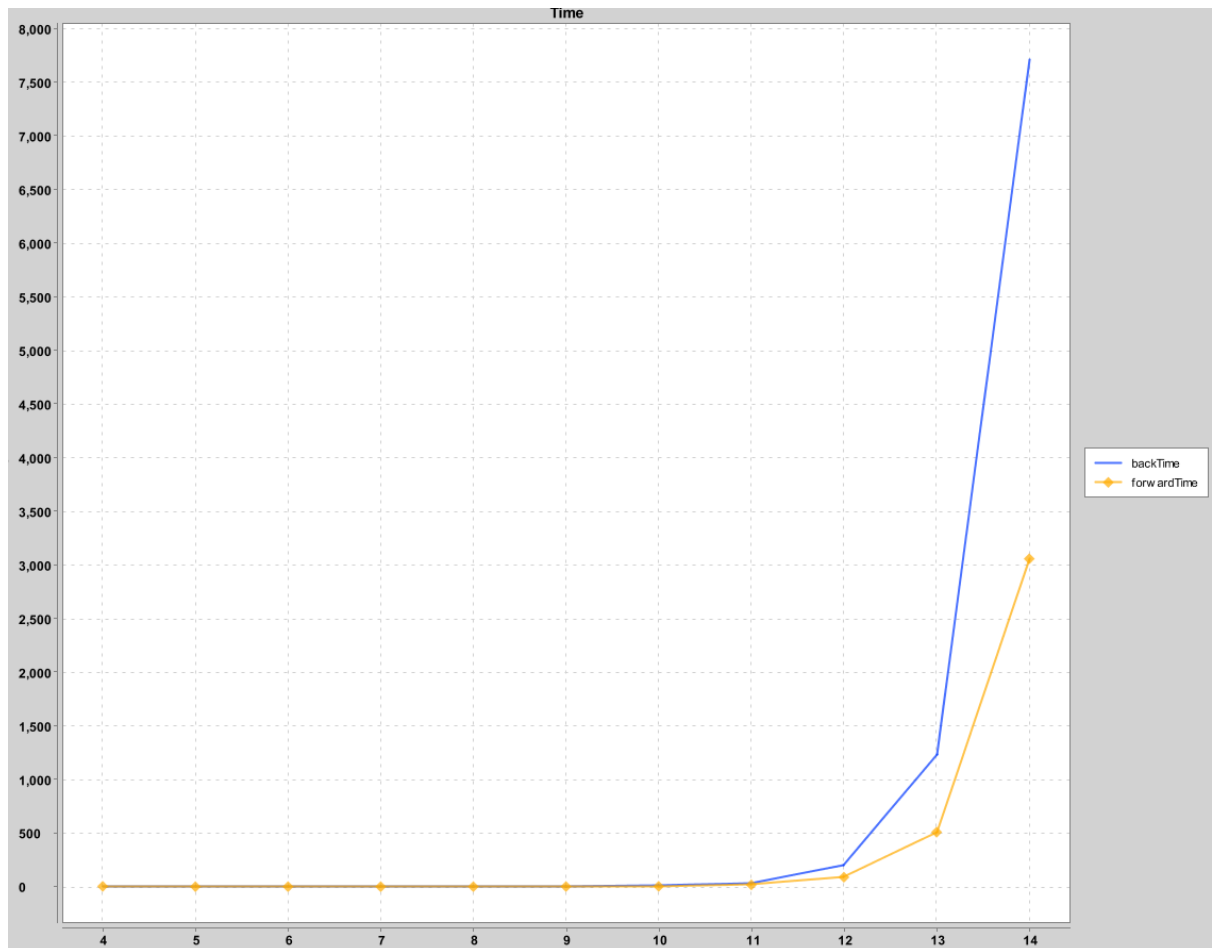
Algorytmy będą różnić się prawdopodobnie:

- Czas trwania algorytmu (z początkową przewagą backtrackingu)
- Liczbą rekurencyjną wywołań algorytmu (z początkową przewagą backtrackingu).

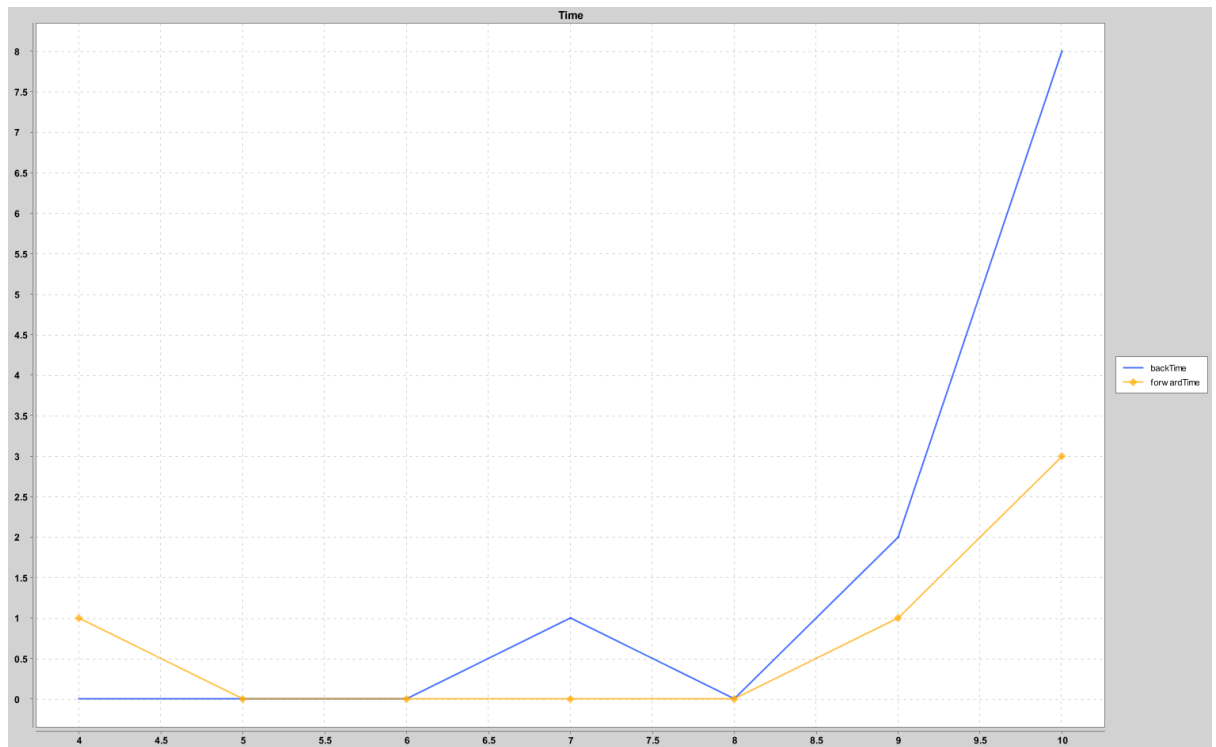
Program został zaimplementowany w Javie

Zbadano następujące dane:

### 1) Czas

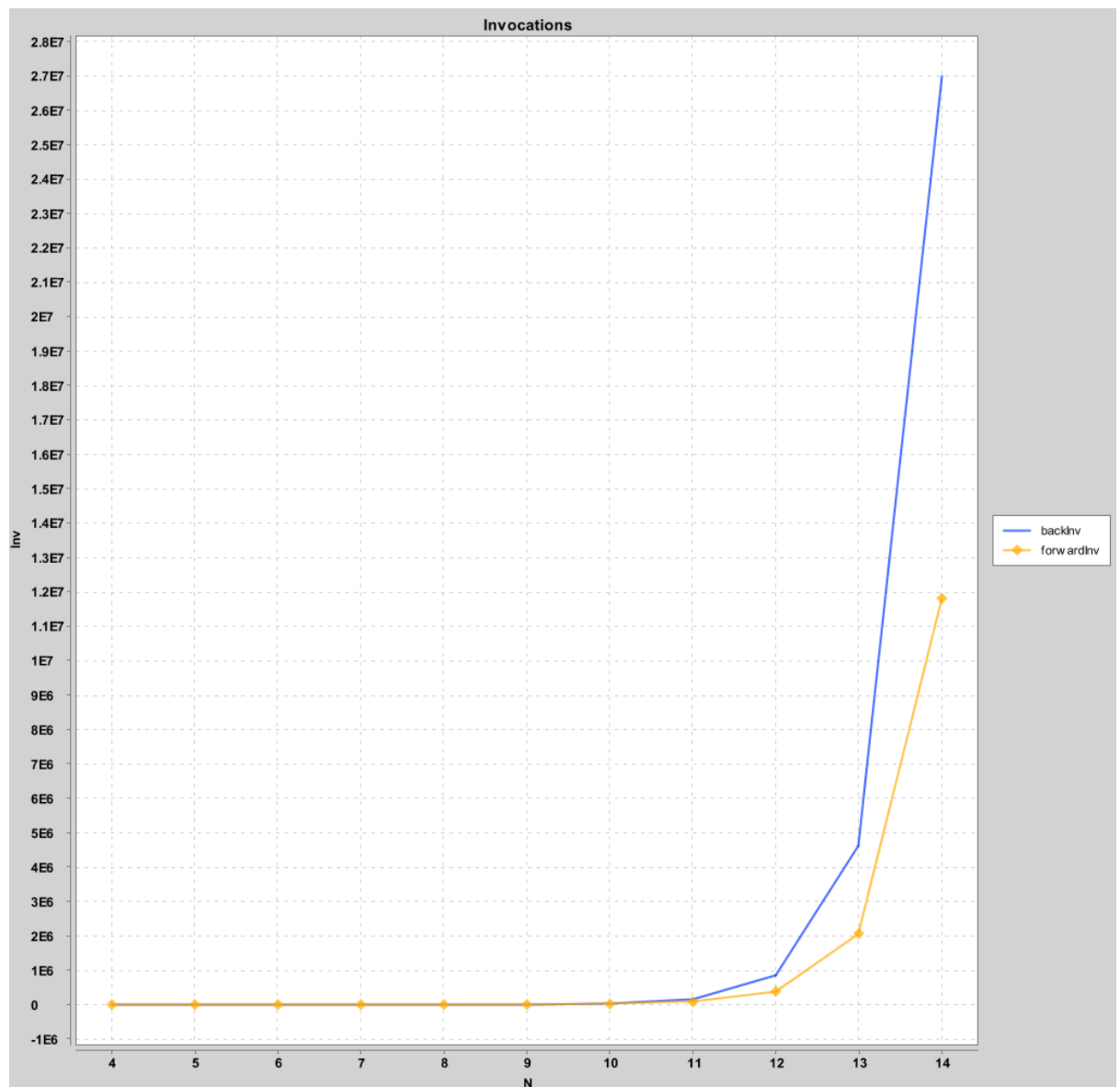


Jak widać, od pewnego poziomu forward checking radzi sobie zdecydowanie lepiej. Sprawdzenie, czy warto generować rozwiązanie i „wchodzić” do danego wężła sprawdza się dobrze, powodując, że dla większych N czas wykonania maleje nawet kilkukrotnie. Aby zobaczyć skalę dla mniejszych N program uruchomiono dla N w zakresie od 4 do 10:



Jednak, z powodu bardzo małych różnic ciężko stwierdzić który z tych algorytmów radzi sobie lepiej dla  $N < 8$ . W teorii, powinien być to backtracking.

- 2) Liczba powrotów (wywołań metody rekurencyjnie)



Dla każdego przypadku liczba powrotów dla forward checkingu jest mniejsza. Oznacza to, że algorytm „mądrzej” wybiera rozwiązania, dzięki czemu przy większym N może on bardziej optymalnie rozwiązywać problem.

## Latin Square

Latin Square jest kolejnym problemem CSP. W tym wypadku naszym zadaniem jest rozmieszczenie liczb z zakresu  $0..N$  na macierzy o rozmiarze  $N \times N$ , w ten sposób, aby każda podmacierz o rozmiarze  $1 \times 1$  była wypełniona, oraz aby żadna z liczb nie powtarzała się w wierszu, lub komórce.

Sytuacja wygląda bardzo podobnie jak w przypadku problemu N-hetmanów. Liczba inwokacji w przypadku algorytmu forward checking jest mniejsza, lecz czas jest większy. Możemy jednak podejrzewać, że czas ten przy większych  $n$  będzie na korzyść Forward Checkingu.

