# Programowanie z ograniczeniami

# Problem N hetmanów

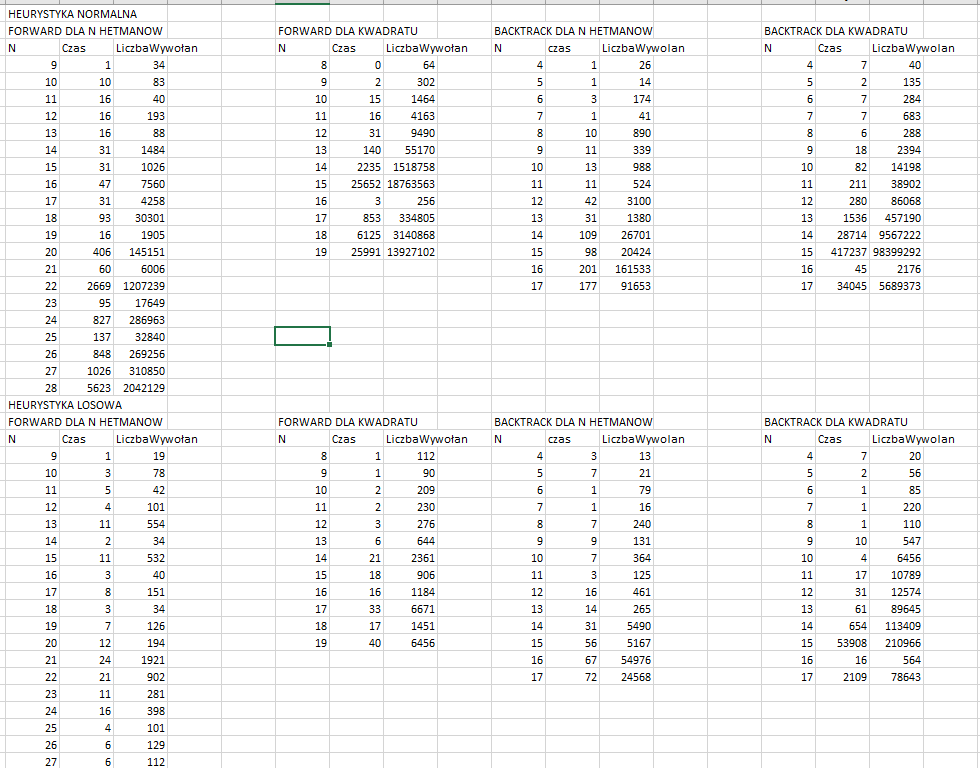
Należy ustawić N hetmanów na tablicy w ten sposób, by każdy z hetmanów nie atakował innego hetmana. Hetman atakuje inną figurę, jeżeli figura znajduje się na przekątnej, bądź na prostej, na której znajduje się hetman.

## Kwadrat łaciński

Kwadrat łaciński rzędu n jest to macierz kwadratowa L: NxN o elementach pochodzących z n-elementowego zbioru liczb S: {1, 2, …, n}, dla której zachodzi własność, taka że w każdym wierszu oraz kolumnie takiej macierzy dany element występuje wyłącznie raz. Kwadrat łaciński można również przedstawić w formie kolorowej kraty o wymiarze NxN, której kolory odpowiadają elementom macierzy spełniając przy tym narzucone ograniczenia. Zadanie sprowadza się do rozwiązania problemu wyznaczenia kwadratu łacińskiego rzędu N: {2, 3, …, k}, rozpoczynając od macierzy zerowej (pustej kraty):

Algorytmy Backtracking implementowałem interacyjnie, natomiast Forward Checking rekurencyjnie. Zastosowałem do tych algorytmów 2 heurystyki wyboru zmiennej : jedna to normalna, czyli po kolei kolejne możliwe wartości a druga to losowa, czyli losuje kolejną możliwą wartość. Warunkiem poprawnego skończenia metody jest znalezienie pierwszego napotkanego dobrego rozwiązania.

Do lepszej interpretacji wykresów zamieszczam zdjęcie z danymi :



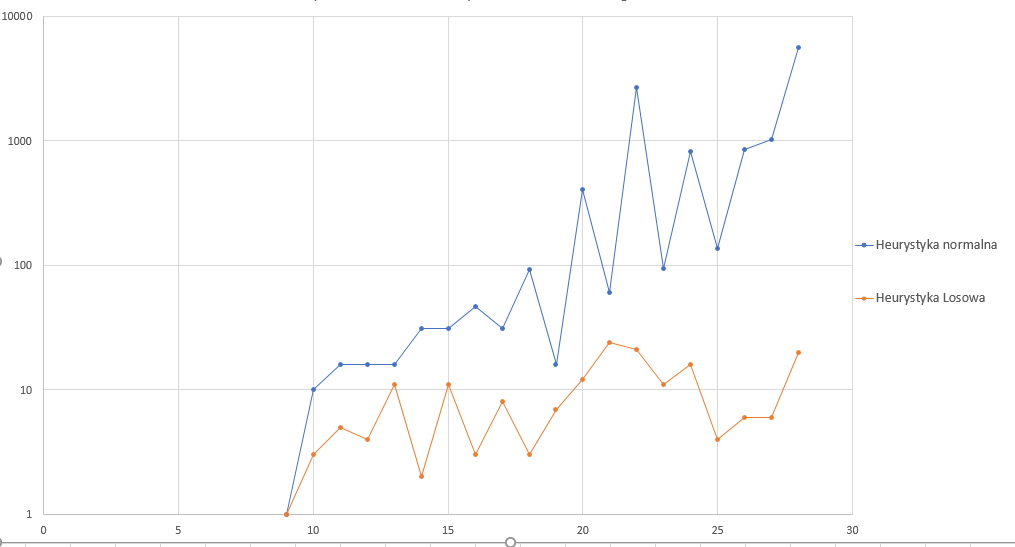
Rys. 1 Dane użyte do wykresów

Wszystkie wykresy są utworzone w skali logarytmicznej o podstawie 10. Czas metod jest podany w milisekundach.

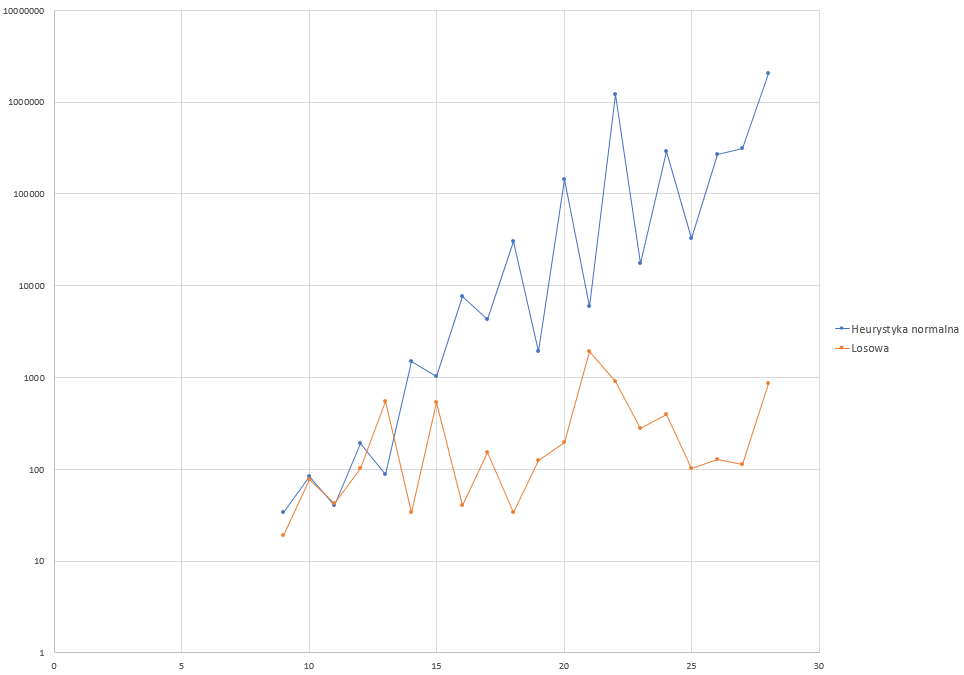
# Algorytm Forward checking

## Dla problemu N hetmanów :

Sprawdziłem 2 zależności: zależność czasu wykonywania całej metody w milisekundach w zależności od liczby N oraz zależność liczby wywołań rekurencji w zależności od N. Przetestowałem wszystko dla dwóch heurystyk: losowej oraz normalnej. Wyniki są następujące :



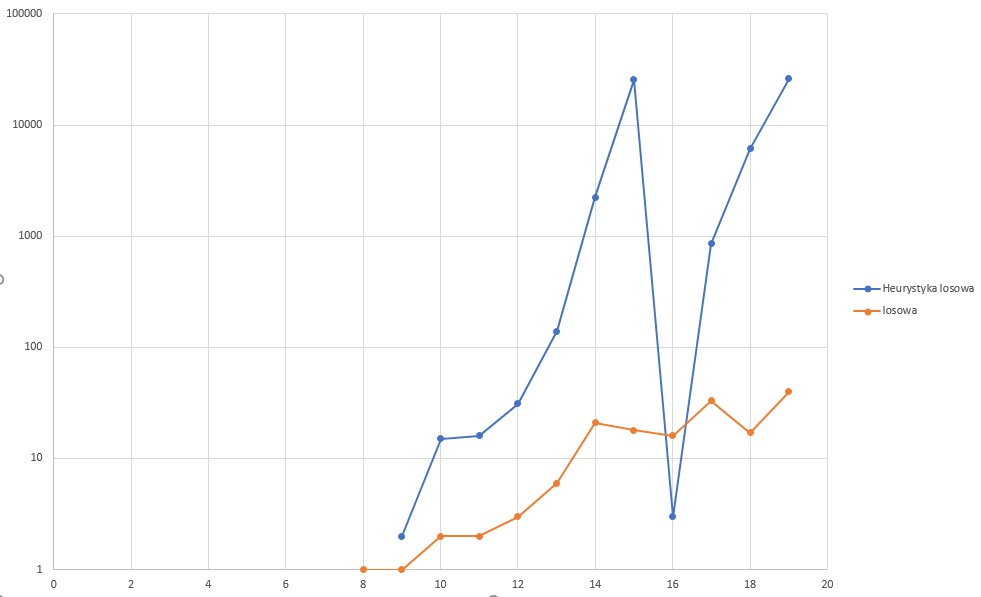
Rys 2. Wykres zależności czasu wykonywania metody w zależności od N



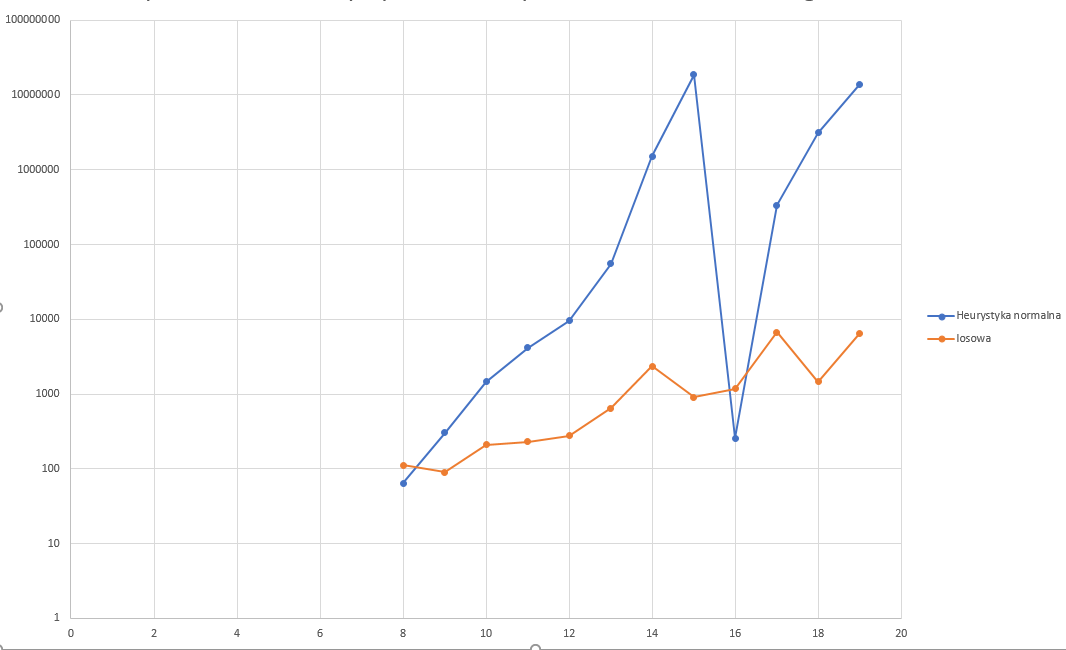
Rys 3. Wykres zależności ilości wywołań rekurencji od liczby N

Jak widać dla N = 22 następuję dziwny wzrost zarówno czasu wykonywania metody jak i liczby wywołań rekurencji. Widzimy również że obie te rzeczy są od siebie zależne, kształty wykresów są niemal identyczne. Widać również, że heurystyka losowa ogromnie polepsza wyniki.

## Dla problemu kwadratu łacińskiego :



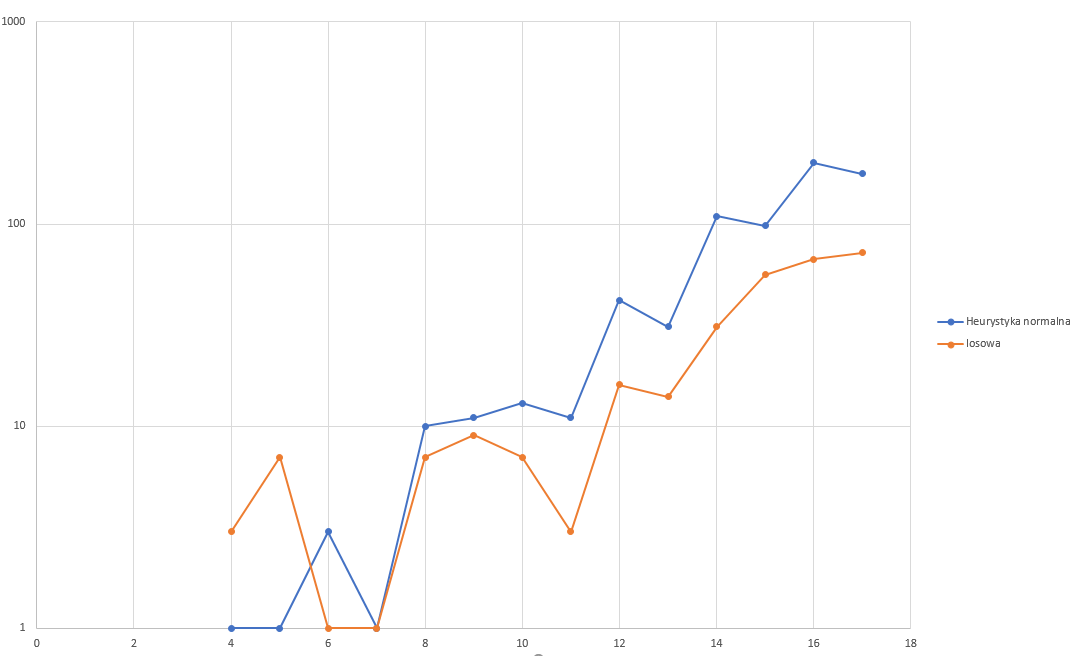
Rys 4. Wykres zależności czasu wykonywania metody w zależności od N

Rys 5. Wykres zależności ilości wywołań rekurencji od liczby N

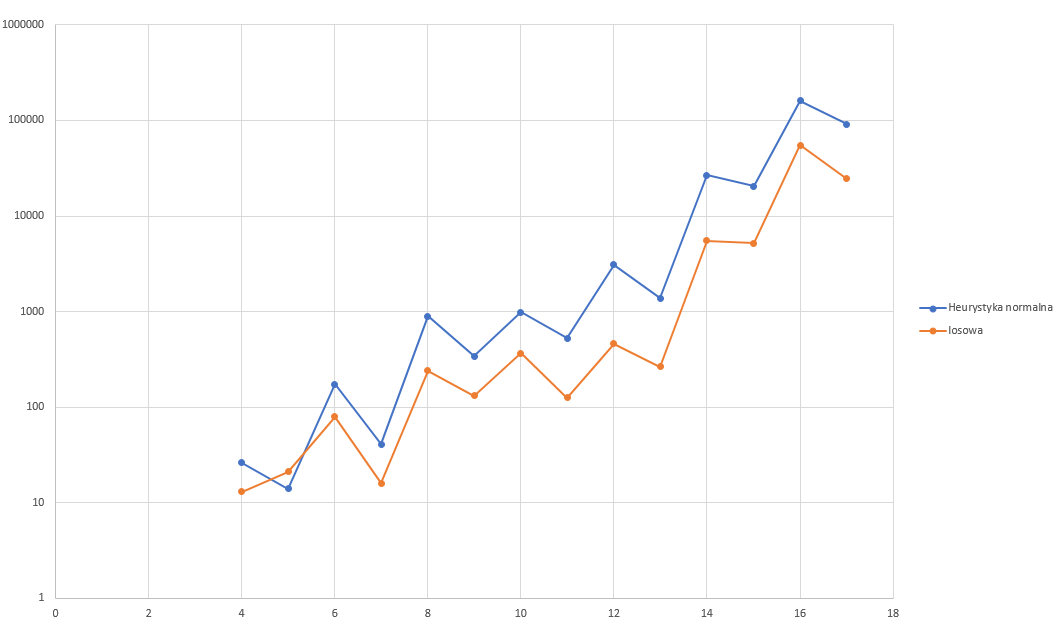
Jak widać na wykresach, również obie te wartości są od siebie bardzo zależne. Kształty wykresów również są niemalże identyczne.

# Algorytm backtracking

## Problem N hetmanów



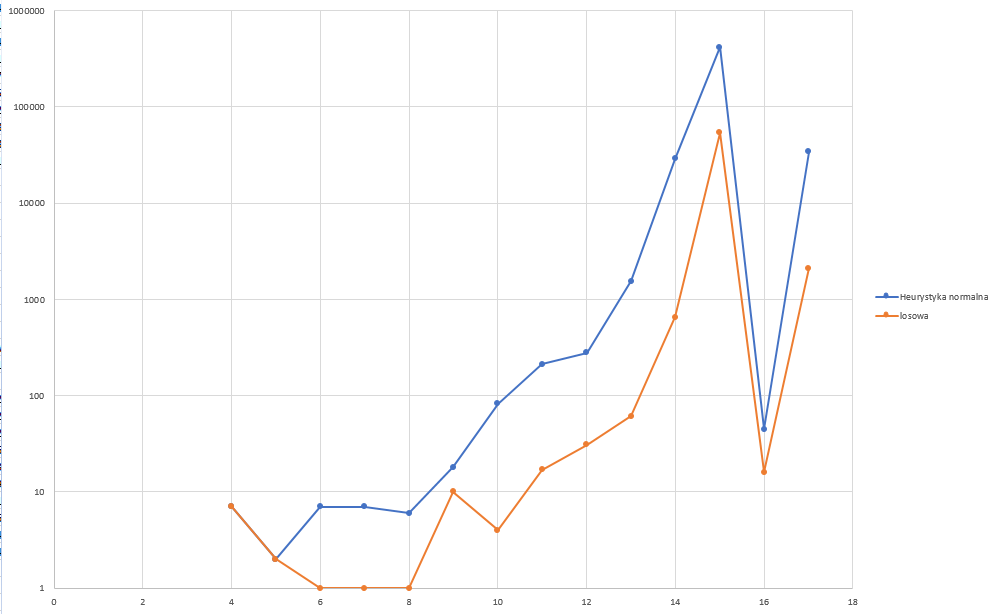
Rys 6. Wykres zależności czasu wykonywania metody w zależności od N



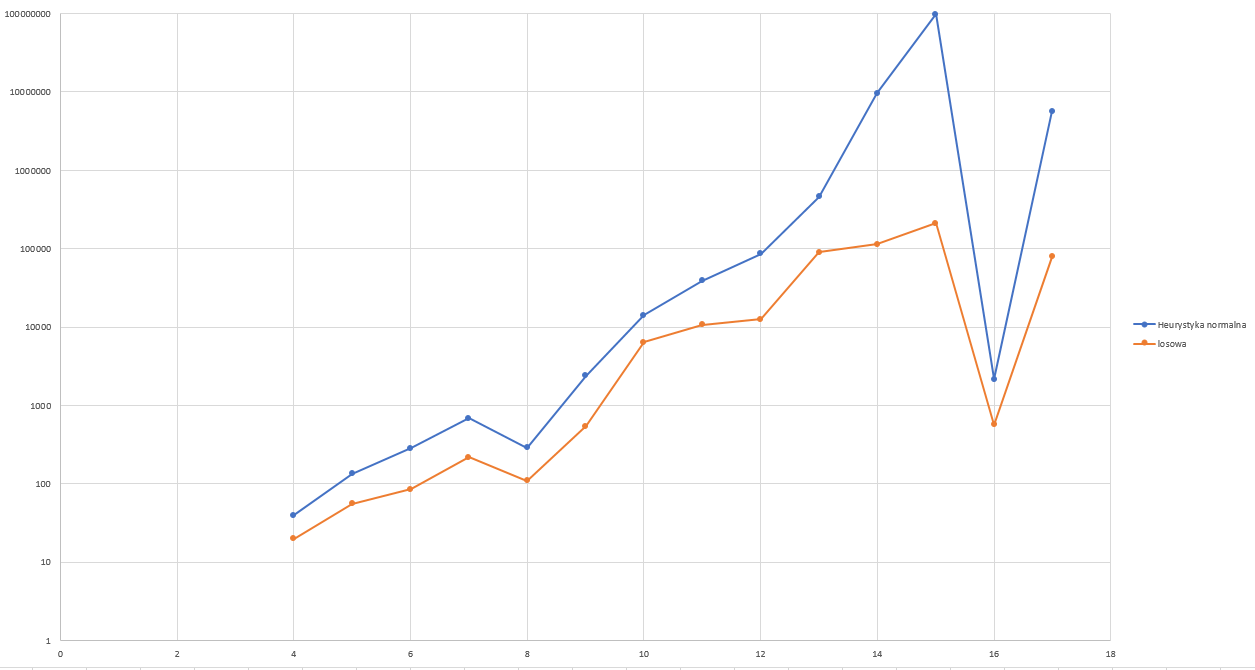
Rys 7. Wykres zależności ilości wywołań iteracji od liczby N

Jak widać na wykresach czas wykonywania metod oraz liczba iteracji algorytmu są od siebie zależne. Można zauważyć nagłe wzrosty czasu oraz ilości wywołań iteracji dla N – 12, 14, 16. Po raz kolejny heurystyka losowa jest o wiele lepsza od normlanej heurystyki.

## Problem kwadratu łacińskiego



Rys 8. Wykres zależności czasu wykonywania metody w zależności od N



Rys 9. Wykres zależności ilości wywołań iteracji od liczby N

Jak widać na wykresach występuje nagły wzrost wartości dla N = 15. I w tym przypadku heurystyka losowa jest o wiele lepsza niż normalna heurystyka.

# Wnoski

Widać, że wybór heurystyki jest bardzo istotny w powyższych algorytmach. Wybierając heurystykę losową zamiast normalnej można otrzymać co najmniej kilkukrotnie lepsze wartości czasów wykonywania metody. Następną rzeczą jaką widać to, że algorytm Forward checking jest wydajniejszy i szybszy od algorytmu Backtracking. Widać, że w większości przypadków wyniki dla heurystyki losowej rosną analogicznie do wyników dla heurystyki normalnej. Wyniki musiałem mierzyć od jakiegoś konkretnego N, ponieważ dla zbyt małej wartości, wyniki dla czasów wychodziły mi zerowe. Na wykresach musiałem użyć skali logarytmicznej, ponieważ w innym wypadku nie dokładnie były by widoczne wyniki.