Kraków, 11.11.2020 r.

Paweł Kruczkiewicz,

Algorytmy geometryczne,

Grupa nr 8 (czwartek 16:15, tydzień B)

Sprawozdanie nr 2

Temat: Otoczka wypukła

Cel ćwiczenia

Sprawdzenie, implementacja, wizualizacja i porównanie dwóch najpopularniejszych algorytmów znajdowania otoczki wypukłej, tj. algorytmów Grahama i Jarvisa.

Wprowadzenie

Otoczką wypukłą zbioru nazywamy wielokąt, w którym (zarówno we wnętrzu jak i na jego bokach) znajdują się wszystkie punkty tego zbioru. Zadaniem algorytmu jest jak najszybsze znalezienie punktów, które tworzą tenże wielokąt.

Omawiane tutaj algorytmy to algorytm Grahama i Jarvisa. Są to najpopularniejsze algorytmy do znajdowania otoczki wypukłej. Dokładne ich działanie przedstawione będzie w poświęconych im punktach w dalszej części niniejszego sprawozdania.

Specyfikacja

Doświadczenie przeprowadzono na 64-bitowym systemie Windows 10 na 4-rdzeniowym procesorze firmy Intel o taktowaniu zegara 2.30 GHz. Kod oraz obliczenia wykonano w Jupyter Notebooku. Wersja pythona to 3.8. Precyzja obliczeń jest ograniczona przez zapis 64-bitowej liczby zmiennoprzecinkowej (odpowiednik typu double w językach C).

Jako wyznacznika użyto samodzielnie zaimplementowanego wyznacznika 3x3 o dokładności $\varepsilon = 10^{-13}$. Wyznacznik ten osiągnął najlepsze wyniki w testach opisanych w poprzednim sprawozdaniu.

Plan doświadczenia

1. Przygotowano program generujący zbiory punktów:
    a. Zawierający n losowo wygenerowanych punktów o współrzędnych z przedziału [a,b]
    b. Zawierający n losowo wygenerowanych punktów na okręgu o środku (a,b) i promieniu R
    c. Zawierający n losowo wygenerowanych punktów leżących na bokach prostokąta o zadanych wierzchołkach.
    d. Zawierający wierzchołki kwadratu, którego jeden wierzchołek znajduje się w punkcie (0,0) oraz generuje losowo po n_1 punktów na bokach na osiach współrzędnych oraz po n_2 punktów na przekątnych.
2. Za pomocą powyższego programu wylosowano dwa typy zbiorów:
    a. Testowe (sprawdzające poprawność), które były zadane w poleceniu ćwiczenia:
        i. zawierający 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100, 100], - Zbiór A
        ii. zawierający 100 losowo wygenerowanych punktów leżących na okręgu o środku (0,0) i promieniu R=10, - Zbiór B

iii. zawierający 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10), - zbiór C
iv. zawierający wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10) oraz punkty wygenerowane losowo w sposób następujący: po 25 punktów na dwóch bokach kwadratu leżących na osiach i po 20 punktów na przekątnych kwadratu. -zbiór D

b. Przeznaczone do wizualizacji – analogiczne do powyższych, jednak ze zmniejszoną liczbą punktów, dzięki czemu można łatwiej przedstawić działanie algorytm (oznaczone dolnym indeksem *v* – np. zbiór A_v).

3. Zwizualizowano utworzone punkty.
4. Zaimplementowano algorytm Grahama i Jarvisa. Każda implementacja zwraca tablicę wierzchołków znajdujących się w otoczce.
5. Napisano program do wizualizacji algorytmu i uruchomiono go dla obu algorytmów na zbiorach do tego przeznaczonych.
6. Sprawdzono sprawność działania algorytmu (tj. ich czas wykonania) na dodatkowo wygenerowanych zbiorach analogicznych do wyżej wymienionych, lecz z większą liczbą punktów.

Oznaczenia graficzne:

niebieski – punkt zbioru (nieaktywny), **zielony** – zawarty w otoczce, **czerwony** – obecnie przetwarzany, **złoty** – znajdujący się na stosie

Algorytm Grahama:

Algorytm Grahama opiera się na użyciu stosu, na którym znajdują się „kandydaci" do wierzchołków otoczki. Poszczególne kroki algorytmu dla zbioru Q wyglądają następująco:

1. Znalezienie punktu p0 o najmniejszej współrzędnej y (w razie remisów – najmniejszego x)
2. Posortowanie punktów wg współrzędnej kątowej względem puntu p0.
3. Usunięcie punktów współliniowych w zbiorze Q.
4. Utworzenie stosu i dodanie punktu p0 oraz 2 pierwszych punktów z posortowanej listy punktów na stos.
5. Dla każdego z pozostałych punktów w liście sprawdzamy jego położenie względem dwóch ostatnich punktów (nazwijmy je s1 i s2) na stosie:
   a. Jeżeli punkt leży na prawo, zdejmujemy punkt s2 ze stosu i sprawdzamy jego względne położenie jeszcze raz. Powtarzamy, dopóki punkt p nie jest po lewej stronie
   b. Jeżeli leży na lewo, dodajemy go do stosu i przechodzimy do kolejnego punktu
6. Na koniec zwracamy stos. Punkty na nim się znajdujące stanowią otoczkę wypukłą.

Schemat działania algorytmu został pokrótce przedstawiony na wykresie poniżej (wykres 1).

Sprawdzenie wszystkich punktów (krok 5) w Q ma czas liniowy, jednak sortowanie w ogólnym przypadku algorytmem quickSort (użytego w implementacji) wynosi O(n*logn), gdzie n to liczebność zbioru Q. Stąd złożoność całego algorytmu wynosi O(nlogn). Ponieważ QuickSort jest szybszy w ogólnym przypadku, jednak działa wolno dla prawie posortowanej tablicy, toteż w zbiorze C i D użyto również alg. Grahama z MergeSortem

Algorytm Jarvisa

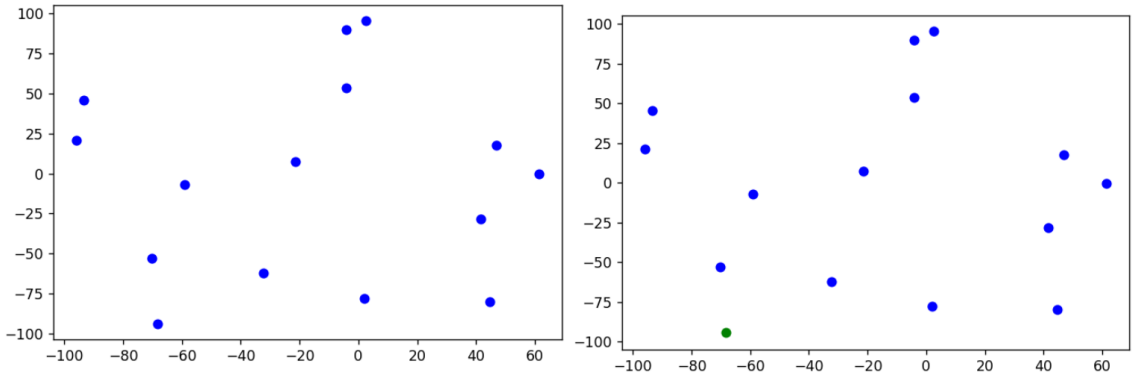
Algorytm Jarvisa korzysta z metody znajdowania otoczki znanej jako **owijanie** (z ang. *Gift-wrapping)*. Można jego działanie zobrazować w prosty sposób – jeżeli nasze wierzchołki to gwoździe na płaszczyźnie, a otoczka to nić, to algorytm Jarvisa znajduje najniżej położony gwóźdź i *naciąga* nić po kolejnych punktach najbardziej oddalonych na prawo.

Formalnie kroki algorytmu można zapisać jako: 1. Znalezienie punktu p0 (jak w alg. Grahama),

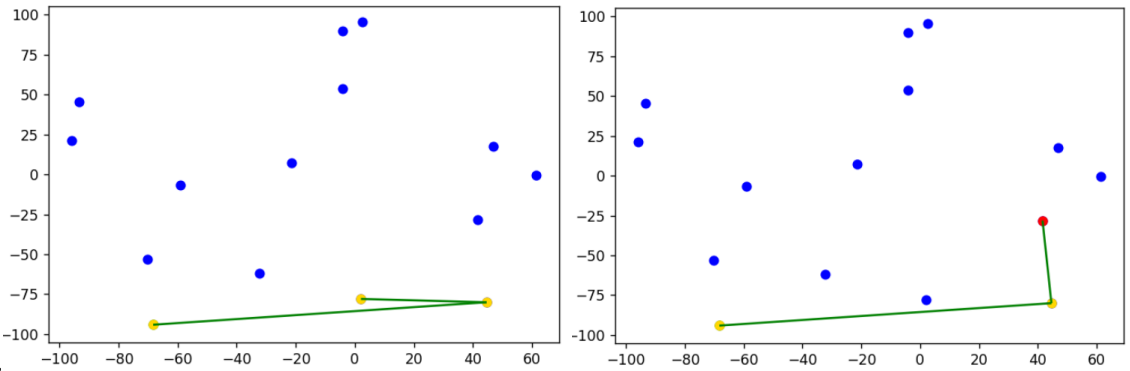
2. Znajduj (w sposób zachłanny) punkty o najmniejszej współrzędnej kątowej względem prostej wyznaczonej przez ten i poprzedni punkt (dla punktu p0 – względem osi OX) i dodaj go do otoczki.

3. Gdy znalezionym punktem będzie p0, zwróć otoczkę.

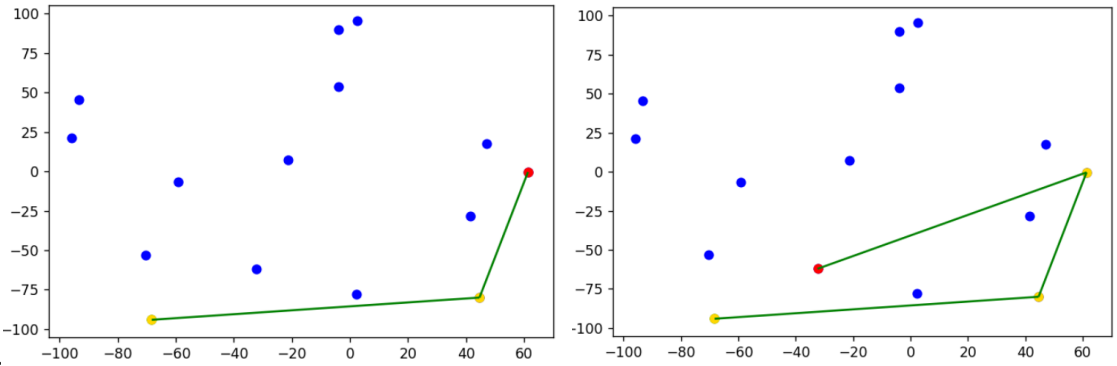
1.



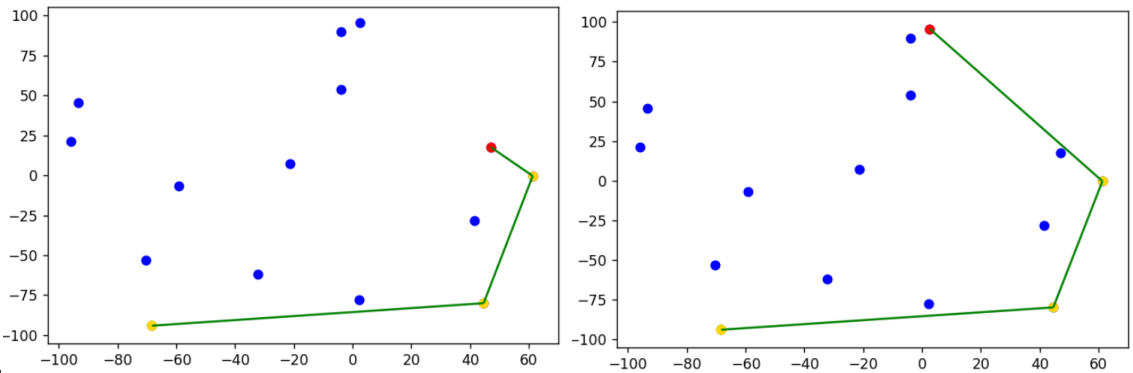
2.



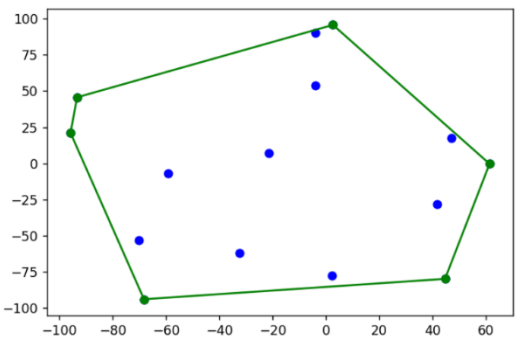
3.



4.



(...)



Wykres 1 - przykład działania algorytmu Grahama

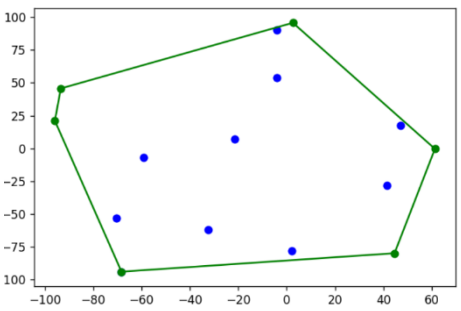
W pierwszym kroku znajdujemy punkt p0 (kolumna 1). Po wcześniejszym posortowaniu i usunięciu punktów współliniowych, algorytm przetwarza każdy punkt, i dodaje go na stos (kolumny 2-4). Ów punkt może zostać z niego ściągnięty (jak w 4. kolumnie powyższego wykresu). Po przetworzeniu wszystkich wierzchołków, algorytm zwraca gotową otoczkę (ostatni obraz)

Algorytm Jarvisa ciąg dalszy

Działanie algorytmu na zbiorze A_v przedstawiono poniżej (wykres 2).

Algorytm Jarvisa ma złożoność O(k*n), gdzie n = |Q| oraz k to liczba wierzchołków w otoczce. Jeżeli jednak zakładamy pesymistycznie, że wielkości otoczki to O(n), to algorytm ma złożoność O(n^2)

1.
2.
3.
4.



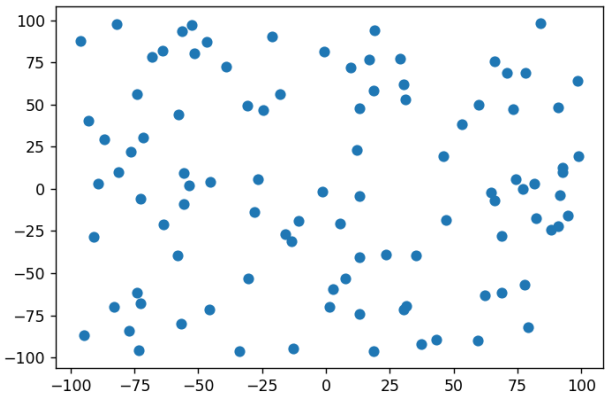
Wykres 2 - przykład działania algorytmu Jarvisa

Podobnie jak w algorytmie Grahama, na początku znajdujemy punkt p0 (kolumna 1). Następnie wyszukujemy punkt o najmniejszej współrzędnej kątowej względem prostej tego i poprzedniego punktu (czyli znajdujemy punkt „najbardziej na prawo" względem prostej stworzonej z przetwarzanego i poprzedniego punktu) i dodajemy go do otoczki (kolumny 2.-4.). Na koniec zwracamy otoczkę (ostatni obraz).

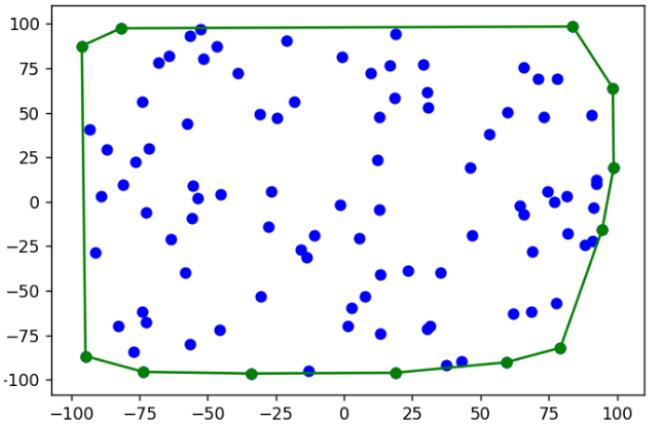
Omówienie wyników ćwiczenia na poszczególnych zbiorach

Zbiór A

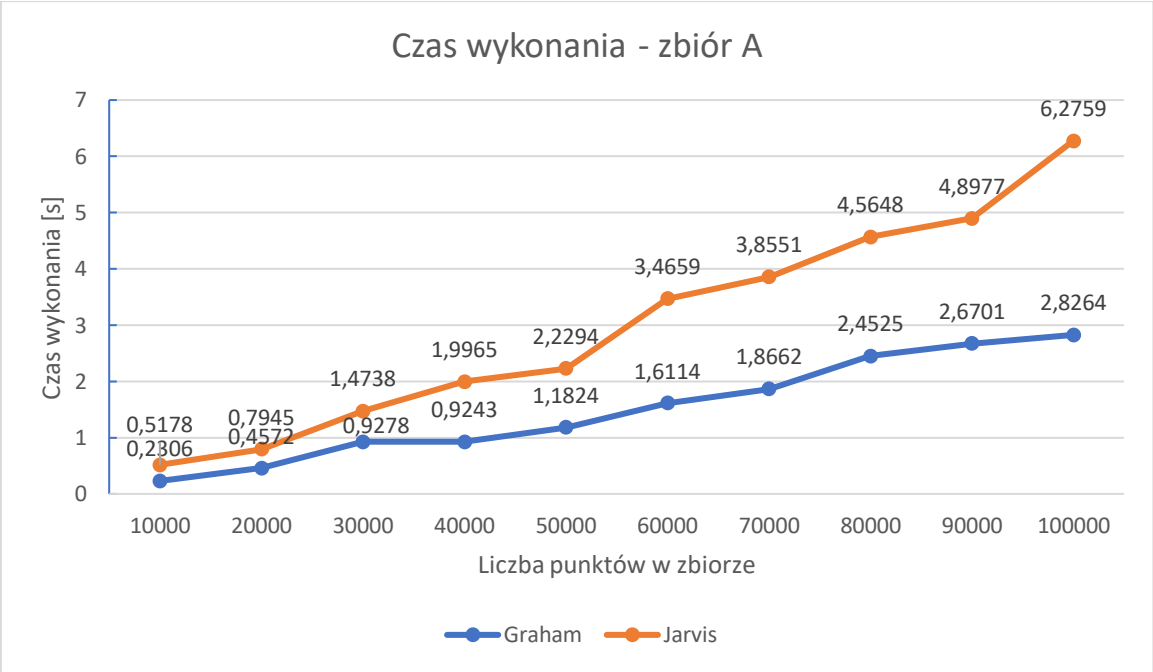
Zbiór A jest zbiorem kontrolnym. Są to rozlosowane na prostokącie punkty. Na poniższych wykresach (3-A i 3-B) przedstawiono wynik działania obu algorytmów.



Wykres 3-A – wizualizacja zbioru A



Wykres 3-B – wizualizacja otoczki wypukłej znalezionej przez alg. Grahama i alg. Jarvisa



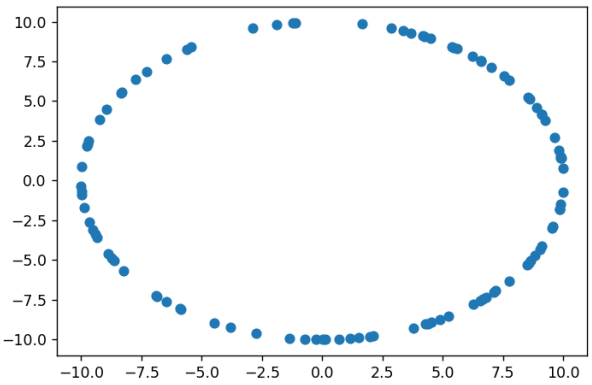
Wykres 4 – czas działania badanych algorytmów dla różnych mocy zbiorów A

Testy czasowe (wykres 4) wskazują na szybsze działanie algorytmu Grahama dla losowo wygenerowanych punktów. Prawdopodobnie jest to spowodowane tym, że liczba punktów w otoczce dla zbioru A jest zazwyczaj większa od logn, co tłumaczyłoby wolniejsze działanie algorytmu Jarvisa względem algorytmu Grahama. Dodatkowo fakt, że szybkość alg. Jarvisa jest zależna od liczby punktów w otoczce sprawia, że dla zbioru o tej samej liczby punktów może wystąpić widoczna różnica w czasie wykonania. Widać to m.in w „skokach” w czasie działania algorytmu (np. między zbiorem 50000 a 60000 lub 90000 a 100000), czego nie obserwujemy na wykresie działania alg. Grahama.

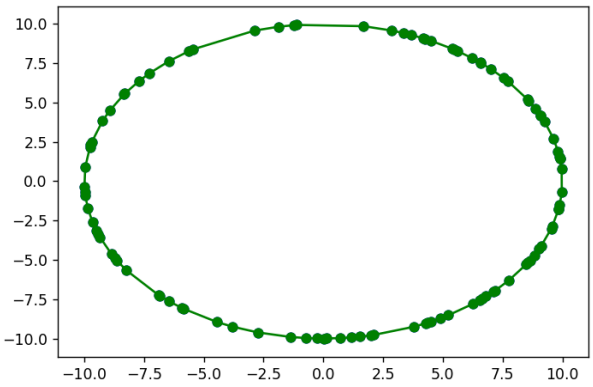
Dokładna wizualizacja działania algorytmu Grahama i Jarvisa na zbiorze A_v została przedstawiona odpowiednio na wykresach 1 i 2.

Zbiór B

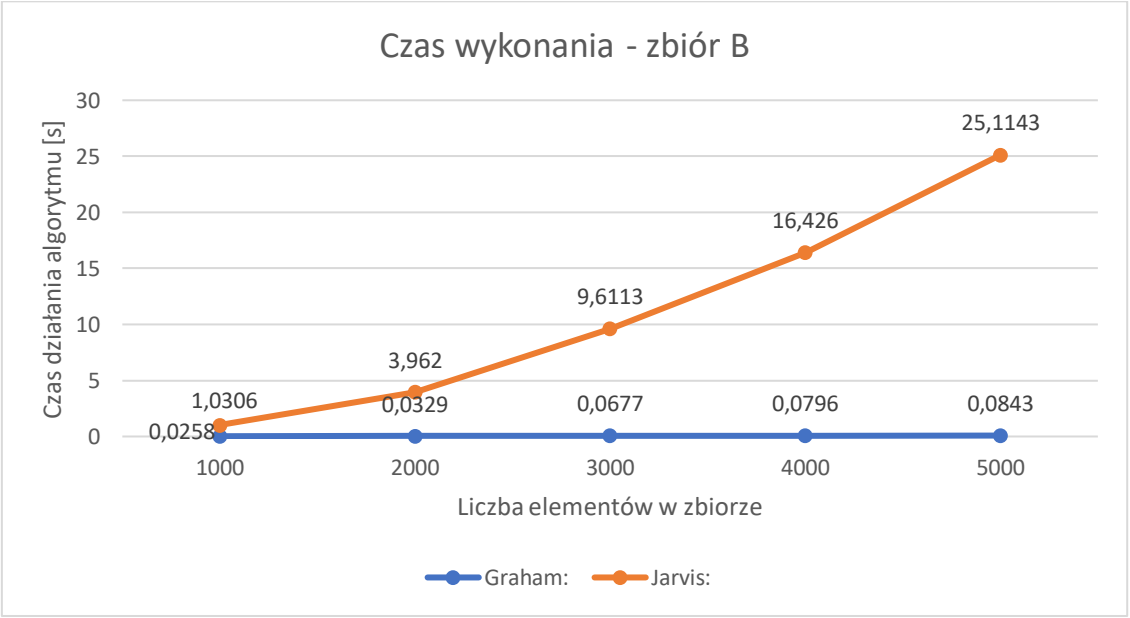
W zbiorze B (wykres 5-A) wszystkie punkty znajdują się na okręgu zatem każdy z punktów powinien zostać przyporządkowany do otoczki(wykres 5-B). Jest to zbiór, dla którego działanie algorytmu Grahama powinno być znacznie szybsze od alg. Jarvisa ze względu na liczbę punktów znajdujących się w otoczce. Dla zbioru B alg. Jarvisa zajmuje czas O(n^2).



Wykres 5-A – wizualizacja zbioru B



Wykres 5-B – wizualizacja otoczki wypukłej

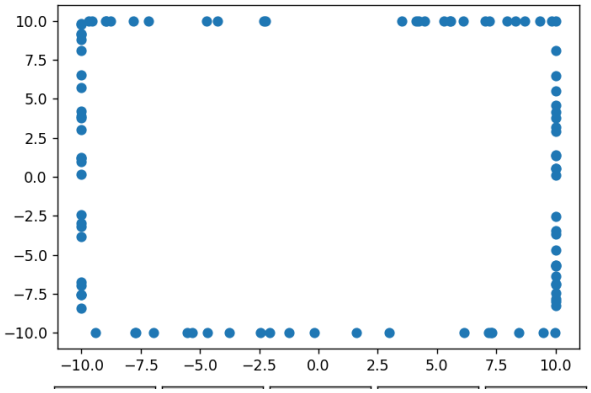


Wykres 6 - *czas działania badanych algorytmów dla różnych mocy zbiorów B*

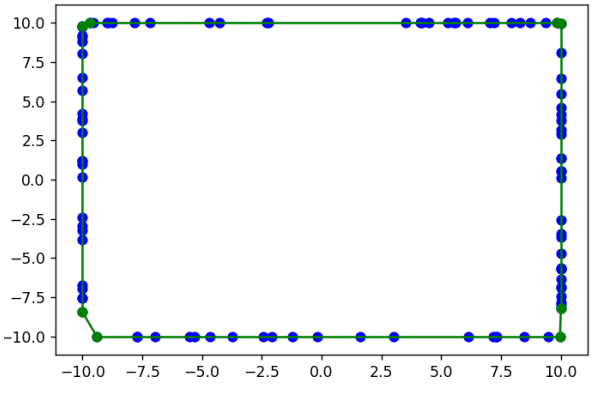
Jak widać na wykresie 6, praktyka popiera teorię – algorytm Jarvisa dla otoczki o liczebności *n* staje się algorytmem $O(n^2)$. Czas dla równych odstępów liczebności elementów rośnie niemal idealnie kwadratowo (w powyższym eksperymencie czasy działania to w zaokrągleniu kolejne kwadraty liczb naturalnych). Dla porównania algorytm Grahama jest rzędy wielkości szybszy - przykładowo dla zbioru 5000 punktów jest szybszy niemal 300 razy.

Zbiór C

Zbiór C to punkty wylosowane na prostokącie o danych wierzchołkach (przedstawiono na wykresie 7-A). Jest to zbiór trudny dla omawianych algorytmów ze względu na dużą liczbę punktów współliniowych. Otoczką wypukłą jest tutaj zbiór 8 punktów – na każdym boku wybierane są te 2 punkty, które znajdują się najbliżej wierzchołków (wykres 7-B). (Same wierzchołki nie należą do zbioru.)

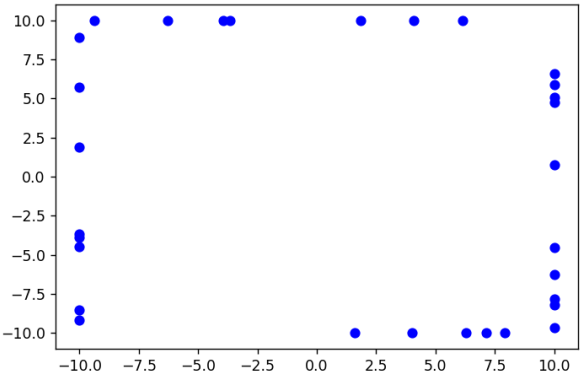


Wykres 7-A – wizualizacja zbioru C

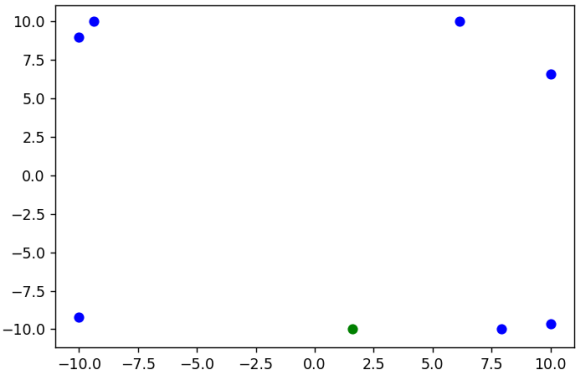


Wykres 7-B – wizualizacja otoczki wypukłej

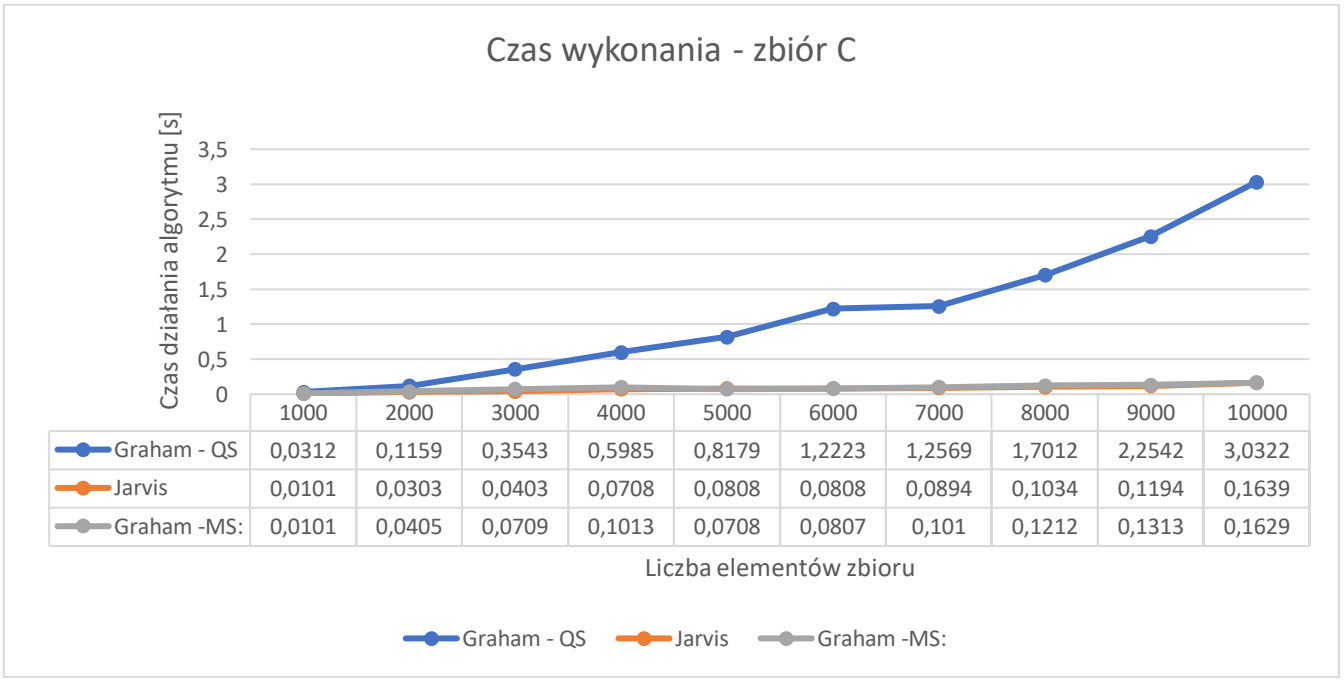
Co ważne – algorytm Grahama już w trzecim kroku usuwa wszystkie punkty współliniowe, czyli dla omawianego zbioru już przed wejściem do głównej pętli algorytmu przetwarzane są jedynie punkty otoczki wypukłej. Działanie tego kroku alg. Grahama dla zbioru C_v przedstawiono na wykresie 8:



Wykres 8- A – pierwotny zbiór punktów



Wykres 8-B – zbiór, na którym operujemy, bez punktów współliniowych



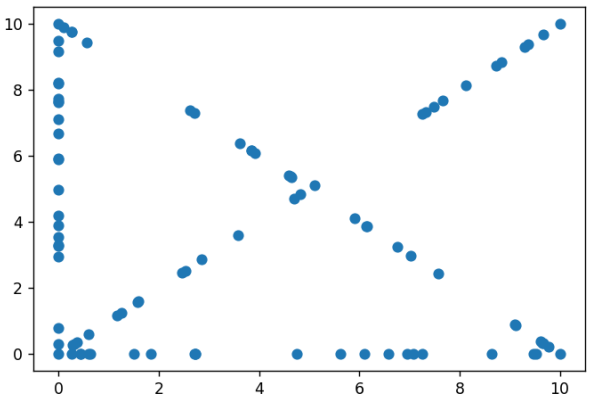
Wykres 9 – czas działania badanych algorytmów dla różnych mocy zbiorów C

Analiza czasu działania (wykres 9) pokazuje wyraźnie, że algorytm Grahama z QuickSortem radzi sobie o wiele gorzej dla zbioru C. Przyczyny można doszukiwać się w tym, że w zbiorze C znajduje się O(n) punktów współliniowych. Z tego powodu Quicksort działa w czasie O(n^2), ponieważ dla grupy punktów współliniowych występuje O(n) zejść rekurencyjnych. Aby zredukować ten problem potrzeba było użyć algorytmu o pesymistycznej złożoności O(nlogn), czyli w tym wypadku mergeSortu. Pozwoliło to zmniejszyć czas działania alg. Grahama niemal do poziomu alg. Jarvisa.

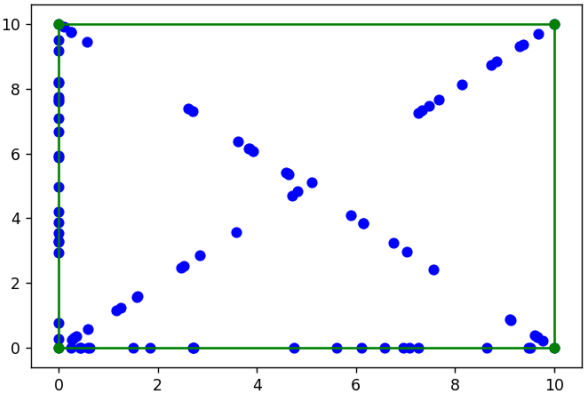
Zauważyć można również, że wiekszość wbudowanych implementacji sortowania opiera się na algorytmie quickSort, więc ów problem może pojawić się w wielu implementacjach algorytmu Grahama.

Zbiór D

Zbiór D składa się z wierzchołków kwadratu, n_1 punktów wylosowanych na 2 jego bokach (lewym i dolnym) oraz n_2 punktów wylosowanych na jego przekatnych (wizualizacja – wykres 10-A). Moc tego zbioru oznaczamy niżej jako $n = n_1 + n_2 + 4$.



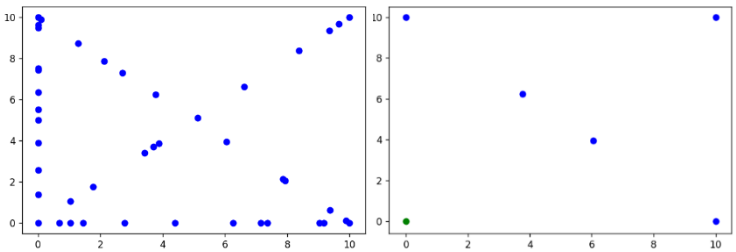
Wykres 10-A – pierwotny zbiór punktów



Wykres 10-B – zbiór, na którym operujemy, bez punktów współliniowych

Zbiór ten sprawia podobne trudności co zbior C. Mamy tu również do czynienia z dużą liczbą punktów współliniowych (ok. ¼ zbioru, czyli O(n)) oraz stałą liczbą (tutaj dokładnie 4) punktów w otoczce (przedstawiona na wykresie 10-B). Pojawia się tutaj jednak dodatkowy problem.

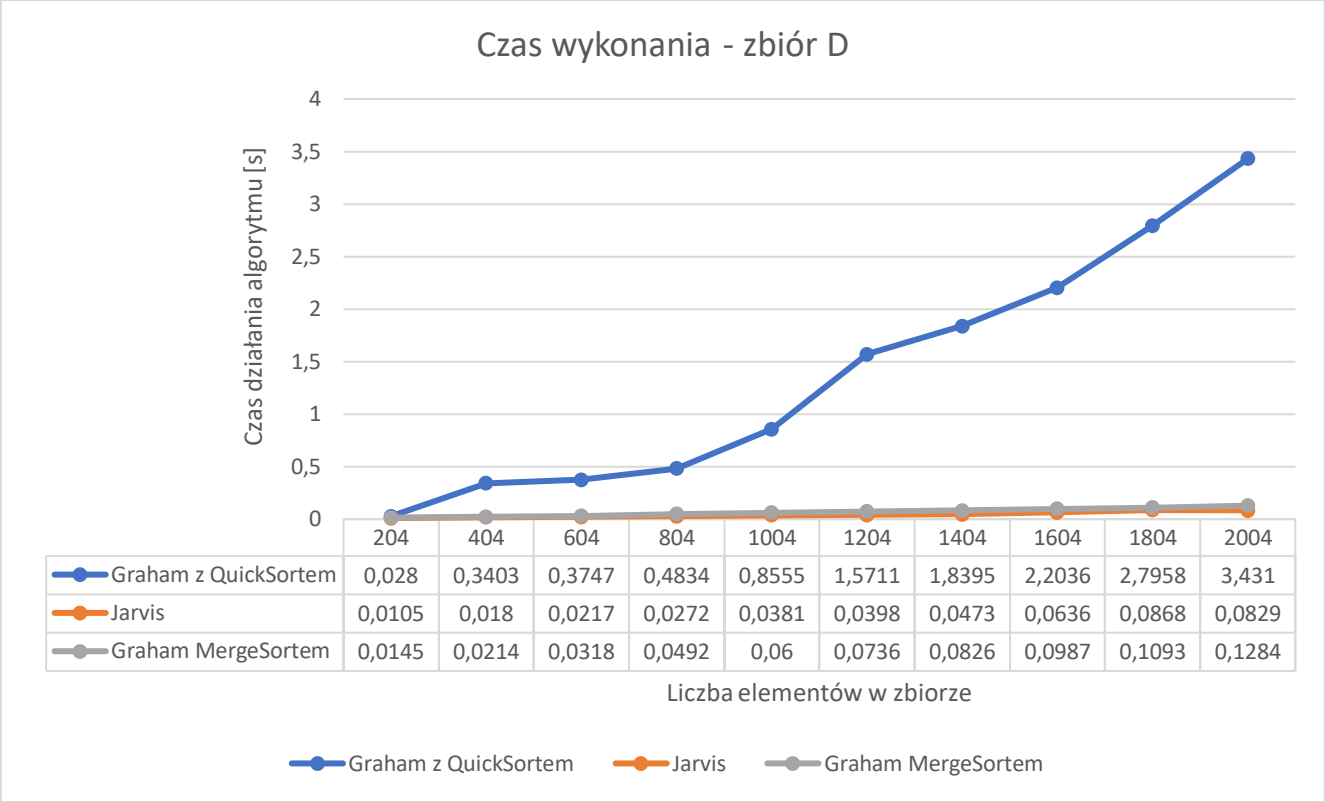
Zaimplementowana przez autora doświadczenia funkcja usuwająca punkty współliniowe polega na sprawdzaniu kolejnych trójek punktów poukładanych już przez algorytm sortujący. Pełny algorytm usuwający punkty współliniowe działałby zawsze w czasie O(n^3)[1], co popsułoby złożoność algorytmu. W efekcie dla zbioru D pozostają 2 punkty znajdujące się na przekątnej, które powinny zostać usunięte (wykres 11). Zastosowany tutaj kompromis nie wychodzi jednak implementacji alogrytmu Grahama na gorsze, przynajmniej pod względem asymptotycznym. Pojedyncze punkty współliniowe nie psują złożoności. Podczas wykonywania zasadniczej pętli algorytmu, pozostałe punkty zostają usunięte ze stosu tak, jakby działo się to dla zbioru A, więc wykonuje się on w czasie liniowo-logarytmicznym.[2]



Wykres 11-A i 11-B – zbiór D przed i po usunięciu punktów współliniowych. Na wykresie B widać dwa punkty we wnętrzu kwadratu, które powinny zostać usunięte

[1] Najprostszy algorytm polegałby na sprawdzeniu każdej trójki punktów, których jest $\binom{n}{3}$

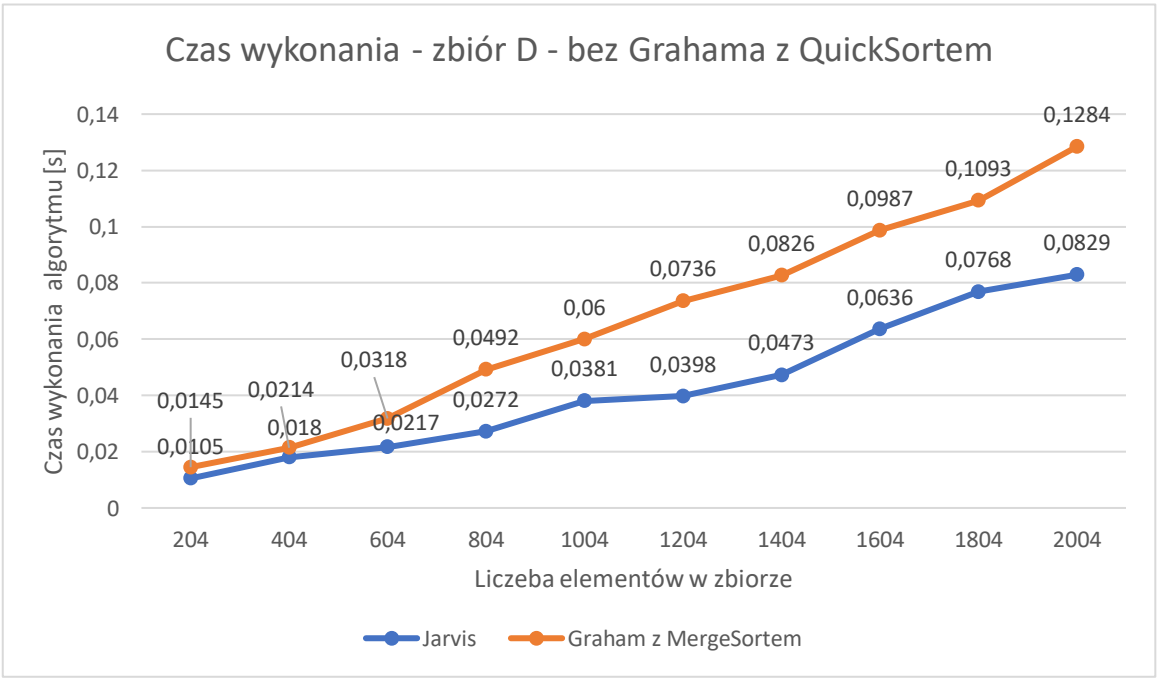
[2] Oczywiście dla zbioru punktów, który byłby „kratą” (czyli składałby się z punktów losowo wygenerowanych na prostych przecinających się np. jak kratki w zeszycie) liczba pozostawionych elementów byłaby duża i uzależniona od liczby przecięć danych linii. Jednak mimo to, błędnie pozostawione punkty współliniowe nie zniszczyłyby złożoności algorytmu.



	204	404	604	804	1004	1204	1404	1604	1804	2004
Graham z QuickSortem	0,028	0,3403	0,3747	0,4834	0,8555	1,5711	1,8395	2,2036	2,7958	3,431
Jarvis	0,0105	0,018	0,0217	0,0272	0,0381	0,0398	0,0473	0,0636	0,0868	0,0829
Graham MergeSortem	0,0145	0,0214	0,0318	0,0492	0,06	0,0736	0,0826	0,0987	0,1093	0,1284

Wykres 12 - czas działania badanych algorytmów dla różnych mocy zbiorów D

Wynik dla zbioru D (wykres 12) jest analogiczny jak dla zbioru C – ze względu na dużą liczbę punktów współliniowych, algorytm Grahama z QuickSortem działa wolno, a alg. Jarvisa działa liniowo ze względu na stałą wielkość otoczki. Warto zauważyć, że algorytm Jarvisa działa tutaj szybciej od algorytmu Grahama z MergeSortem nawet dla niewielkich zbiorów (wykres 13).



Wykres 13 - czas działania badanych algorytmów dla różnych mocy zbiorów D – tylko alg. Graham z MergeSortem i alg. Jarvisa

Wnioski

Opisywane doświadczenie pokazuje, że każdy z dwóch omawianych algorytmów ma swoje słabe i mocne strony. W przypadku punktów losowo wygenerowanych we wnętrzu prostokąta oraz zbiorów o dużej liczbie punktów w otoczce, algorytm Grahama jest lepszy od alg. Jarvisa. Natomiast dla zbiorów, gdzie liczba punktów w otoczce jest stała, algorytm Jarvisa radzi sobie lepiej.

Dodatkowo, ponieważ alg. Grahama wykorzystuje sortowanie, może łatwo osiągnąć czas O(n^2) dla dużej liczby współliniowych punktów (jak dzieje się w przypadku użycia quickSorta jako funkcji pomocniczej). Dobrym wyjściem z takiej sytuacji jest użycie algorytmu sortowania o pesymistycznej złożoności O(nlogn) (np. heapSort lub mergeSort)