Raport - praca domowa nr 2

Tomasz Suchodolski

13 maja 2019

Wstęp teoretyczny

Zadanie analizy skupień (ang. cluster analysis) polega na tym, że mamy n punktów pewnej przestrzeni. Wśród tych punktów znaleźć taki podział (podział w rozumieniu matematycznym) zbioru punktów na takie niepuste, parami rozłączne podzbiory aby punkty wewnątrz danego podzbioru punkty były maksymalnie *podobne* do siebie, jednocześnie punkty leżące w róznych zbiorach były od siebie maksymalnie *odmienne*.

W naszym przypadku będziemy badali punkty znajdujące się w k-wymiarowej przestrzeni euklidesowej. Przez *podobieństwo* punktów do siebie będziemy rozumieli odległość euklidesową punktów od siebie.

Zbiory benchmarkowe

Na repozytorium przedmiotu na GitHubie znajdowało się kilkadziesiąt zbiorów benchmarkowych wraz z wzorcowymi podziałami tych zbiorów. Ponadto przygotowałem 3 własne zbiory wraz z etykietami referencyjnymi (szczegóły w pliku testy.pdf).

Porównanie danych z danymi referencyjnymi

Aby sprawdzić, czy obliczone przez algorytmy dane są podobne do danych referencyjnych posługiwać się będę dwoma indeksami **“indeksem Fowlkesa–Mallowsa”** oraz **“skorygowanym indeksem Randa”**, które zwracają 1 jeśli otrzymane k-podziały są identyczne i wartości odpowiednio mniejsze jeśli podziały się różnią.

Badane algorytmy

W stworzeniu rankingów algorytmów wykorzysam następujące algorytmy

1. Metodą własnoręcznie napisaną zgodnie z algorytmem z treści zadania (szczegóły w testy.pdf)
2. wszystkie algorytmy hierarchiczne z funkcji **hclust()** czyli:
    - “ward.D”
    - “ward.D2”
    - “single”
    - “complete”
    - “mcquitty”
    - “average”
    - “median”
    - “centroid”
3. algorytm **Genie** z pakietu *genie*
4. algorytm **pcm** z pakietu *cluster*

Analiza

Porównanie ogólne

Analizę rozpocznę od przeanalizowania wyników dla wszytkich wyżej wymienionych metod, dla 46 zbiorów danych (43 z repozytorium oraz 3 własne). Celem tej analizy jest znalezienie algorytmów, które dla tych zbiorów działają *najlepiej*, oraz takich, które działają *najgorzej*.

Zbiory benchmarkowe

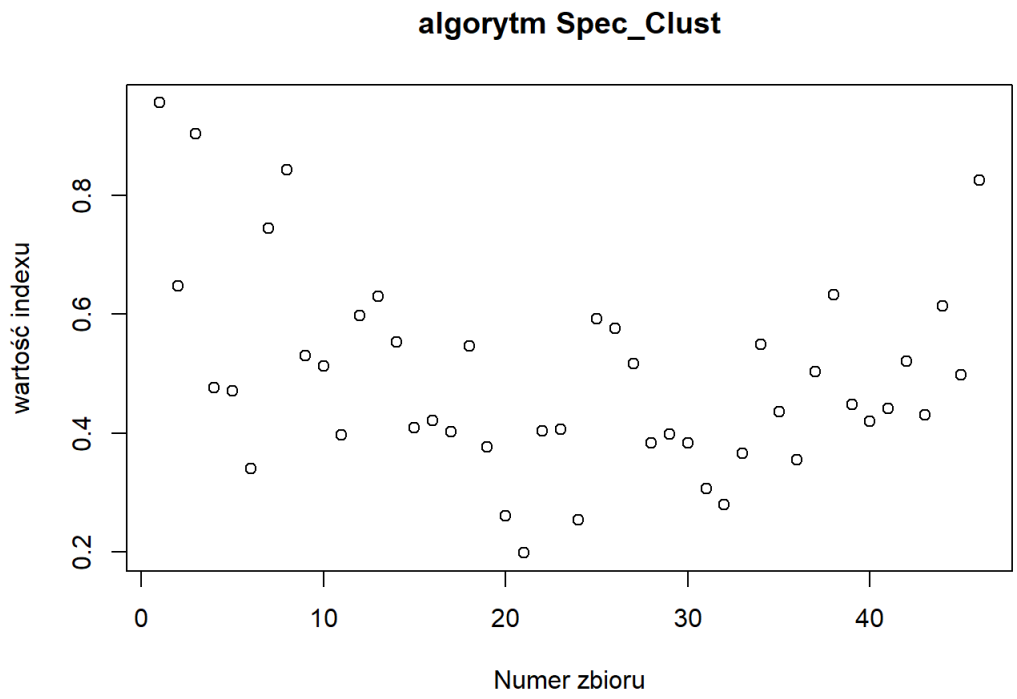
Wszędzie, gdzie na wykresach występowały będą zbiory benchmarkowe w liczbie 46, będą występować one odnosić się do zbiorów, umieszczonych w plikach o poniżej wypisanych nazwach w następującej kolejności:

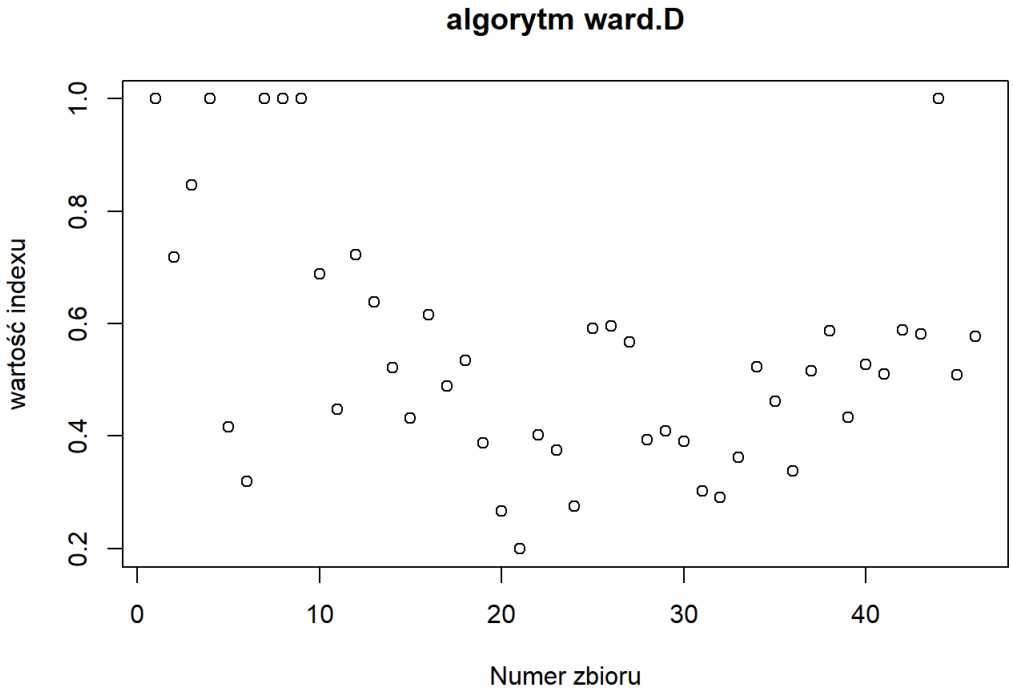
```
##  [1] "fcps/atom"         "fcps/chainlink"    "fcps/engytime"
##  [4] "fcps/hepta"        "fcps/lsun"         "fcps/target"
##  [7] "fcps/tetra"        "fcps/twodiamonds"  "fcps/wingnut"
## [10] "graves/dense"      "graves/fuzzyx"     "graves/line"
## [13] "graves/parabolic"  "graves/ring"       "graves/zigzag"
## [16] "other/iris"        "other/iris5"       "other/square"
## [19] "sipu/a1"           "sipu/a2"           "sipu/a3"
## [22] "sipu/aggregation"  "sipu/compound"     "sipu/d31"
## [25] "sipu/flame"        "sipu/jain"         "sipu/pathbased"
## [28] "sipu/r15"          "sipu/s1"           "sipu/s2"
## [31] "sipu/s3"           "sipu/s4"           "sipu/spiral"
## [34] "sipu/unbalance"    "wut/cross"         "wut/smile"
## [37] "wut/twosplashes"   "wut/x1"            "wut/x2"
## [40] "wut/x3"            "wut/z1"            "wut/z2"
## [43] "wut/z3"            "set1"              "set2"
## [46] "set3"
```

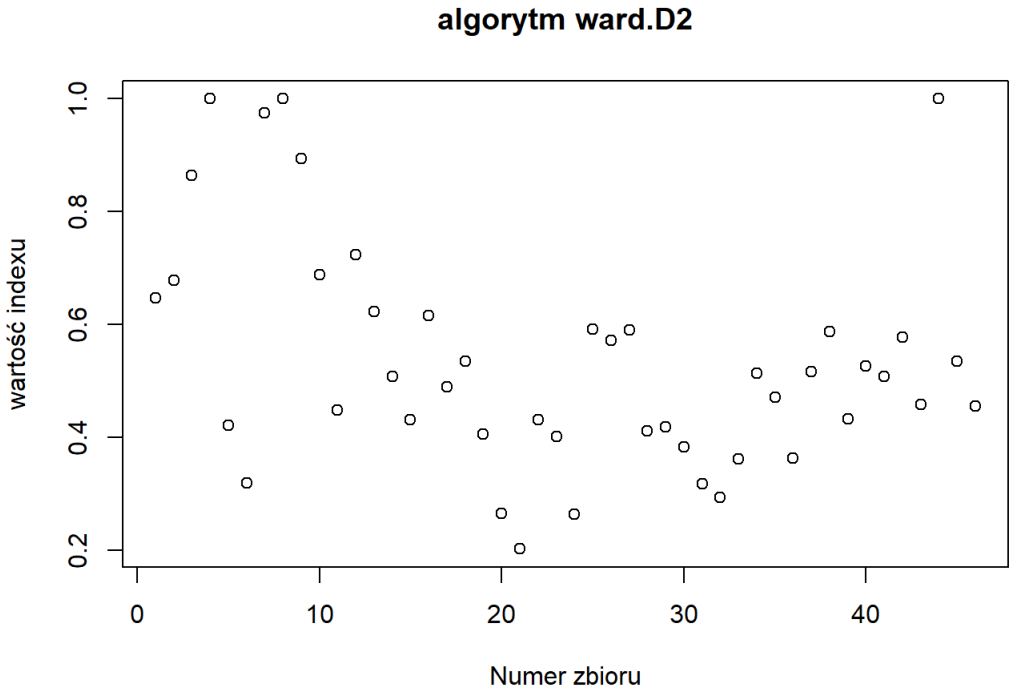
Wykresy

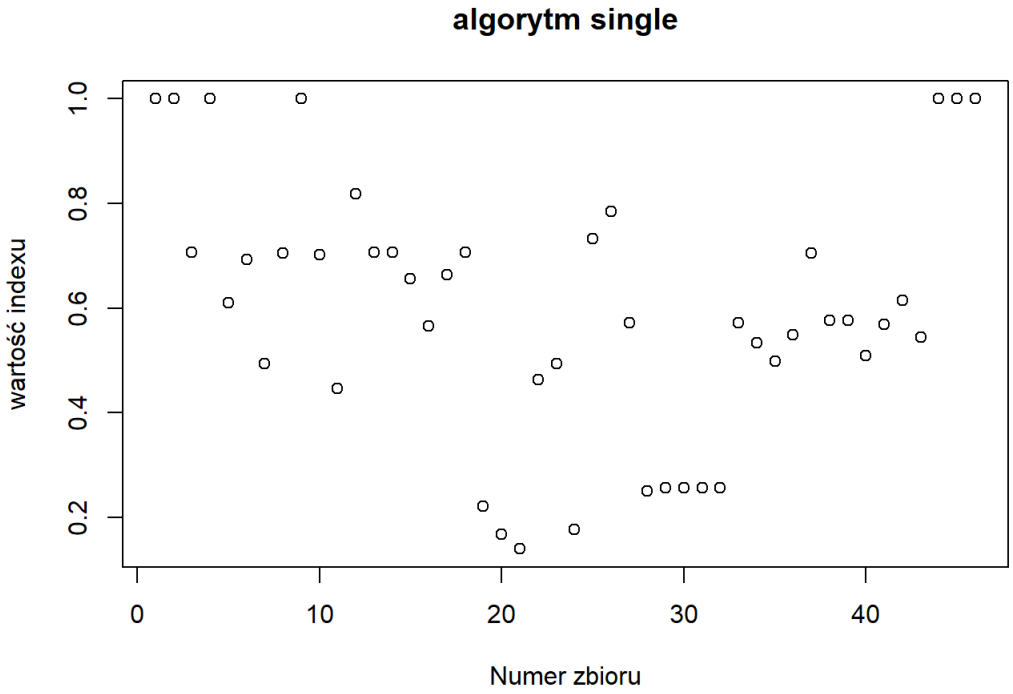
Poniżej zamieszczam wykesy zależności dla wszystkich badanych metod. Przedstawiają one wartość danego indeksu w zależności od numeru zbioru benchmarkowego.

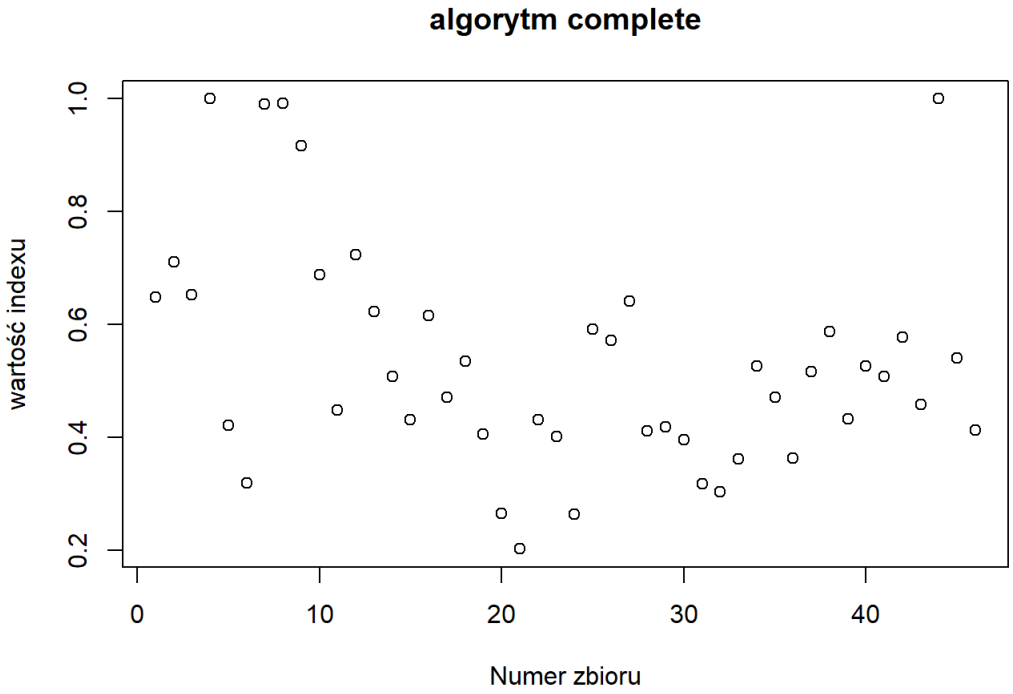
indeks Fowlkesa–Mallowsa

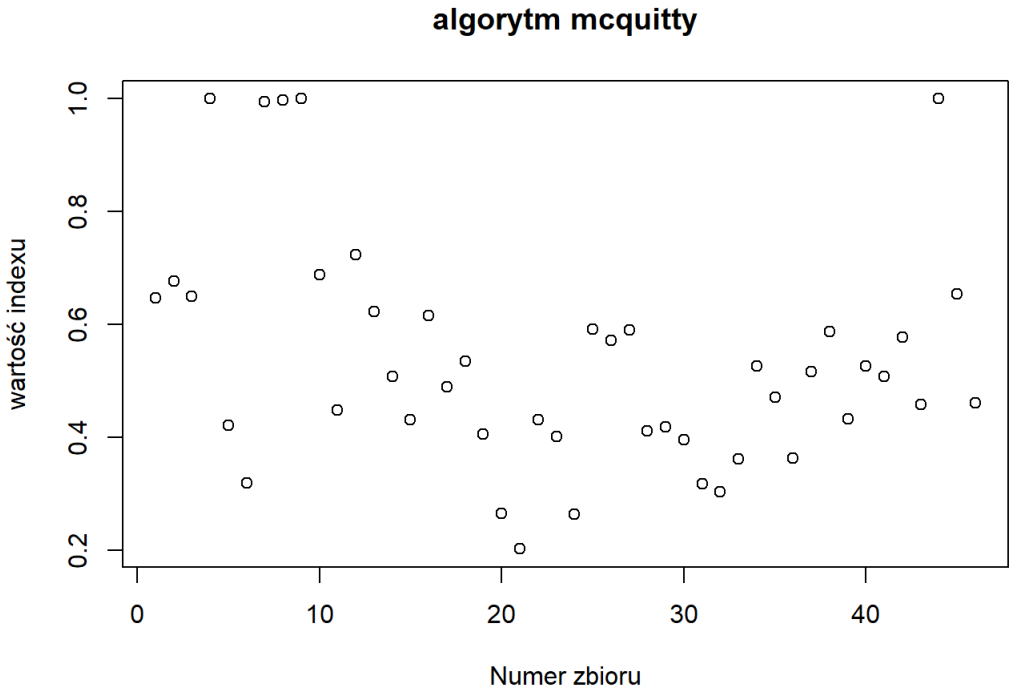


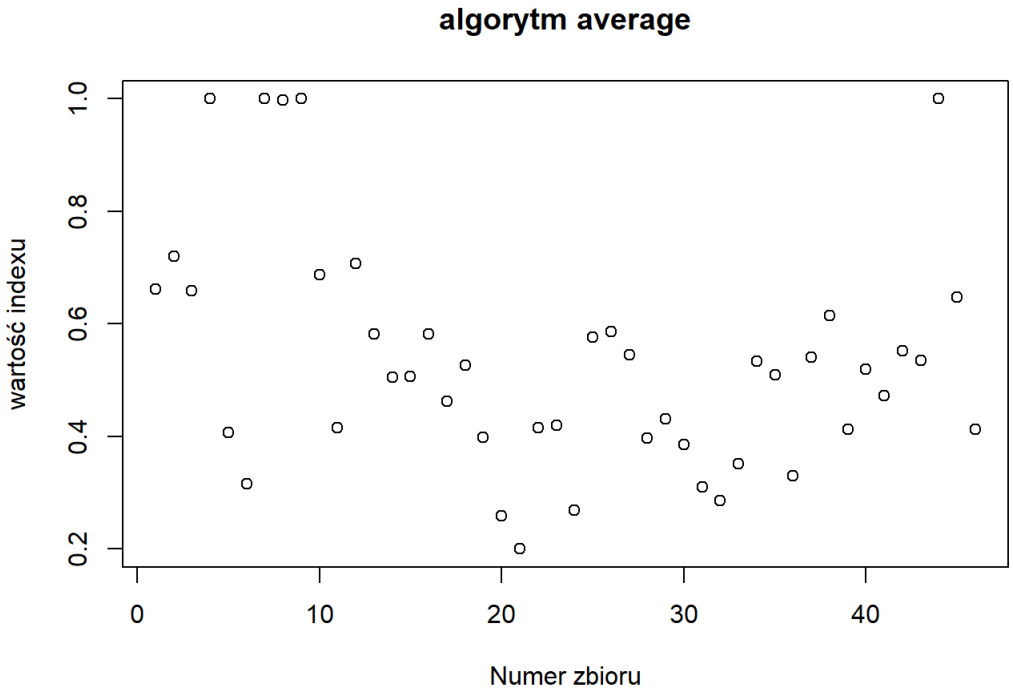

algorytm ward.D
wartość indexu
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

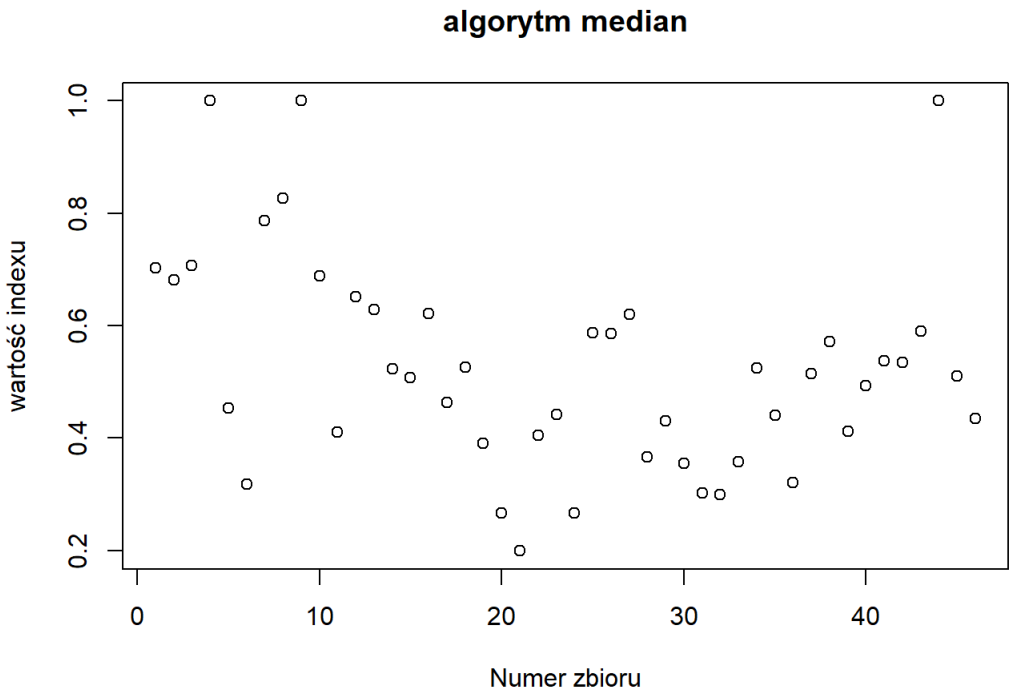

algorytm ward.D2
wartość indexu
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

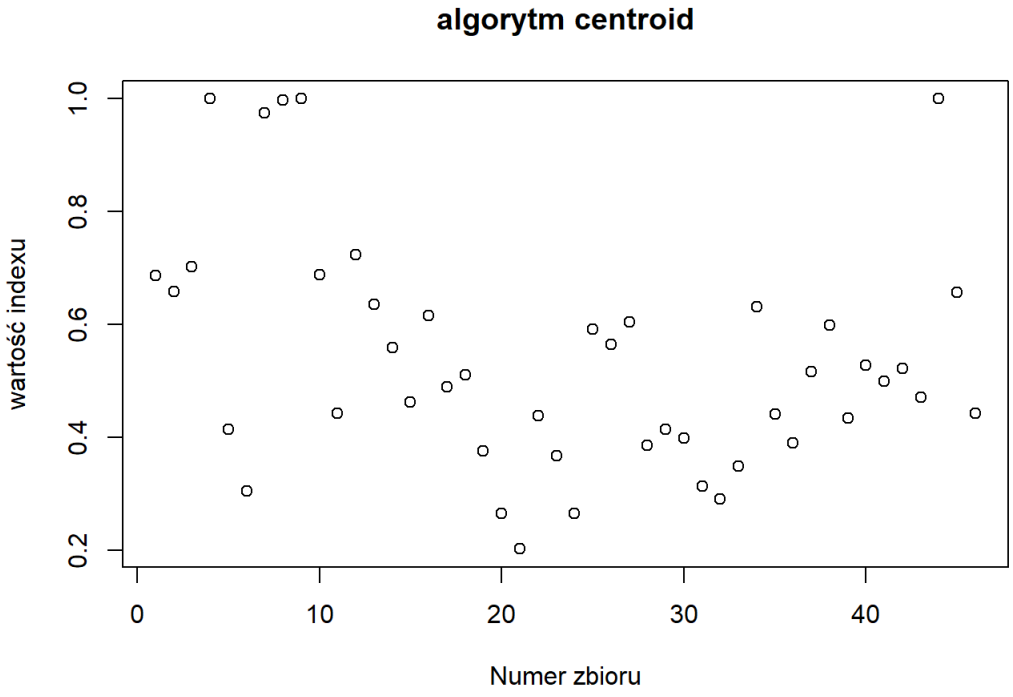

algorytm single
wartość indexu
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

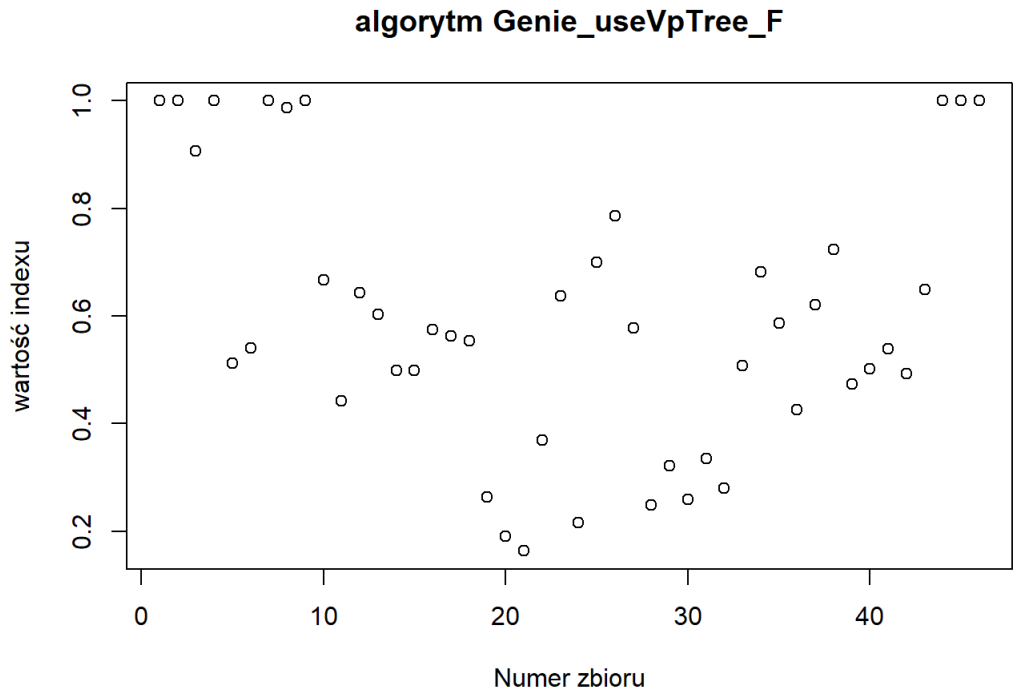

algorytm complete
wartość indexu
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

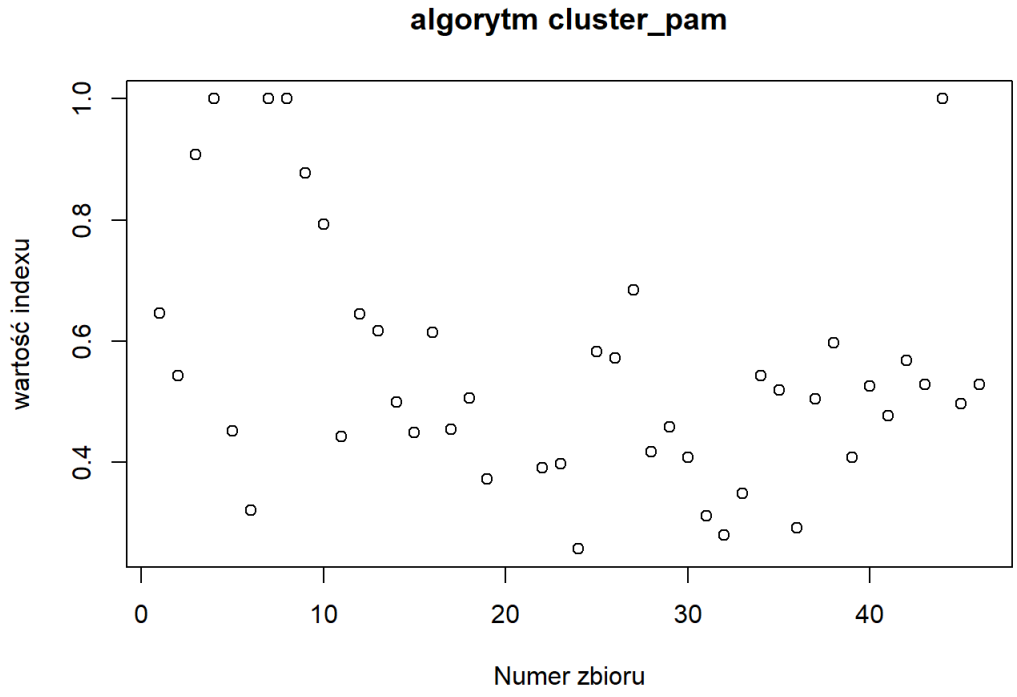

algorytm mcquitty
wartość indexu
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru


algorytm average
wartość indexu
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

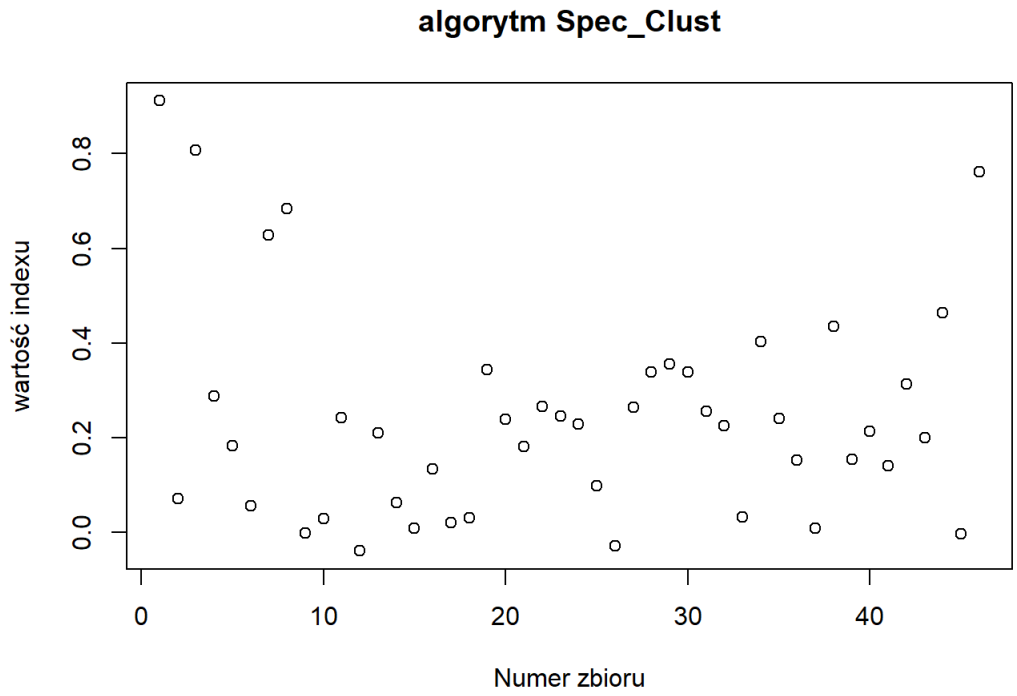

algorytm median
wartość indexu
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

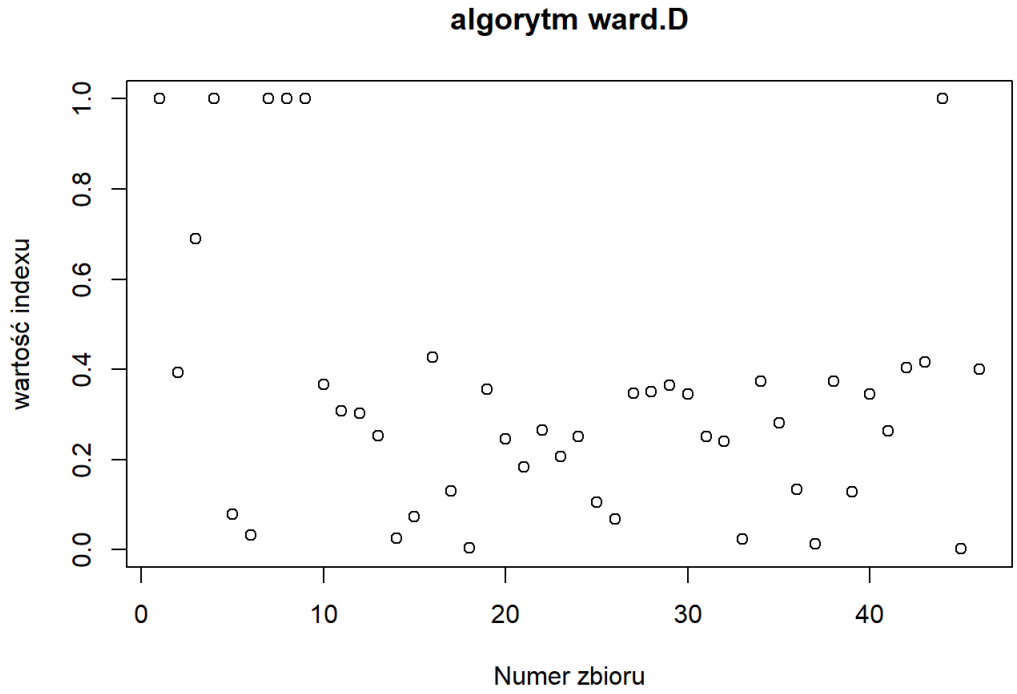

algorytm centroid
wartość indexu
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

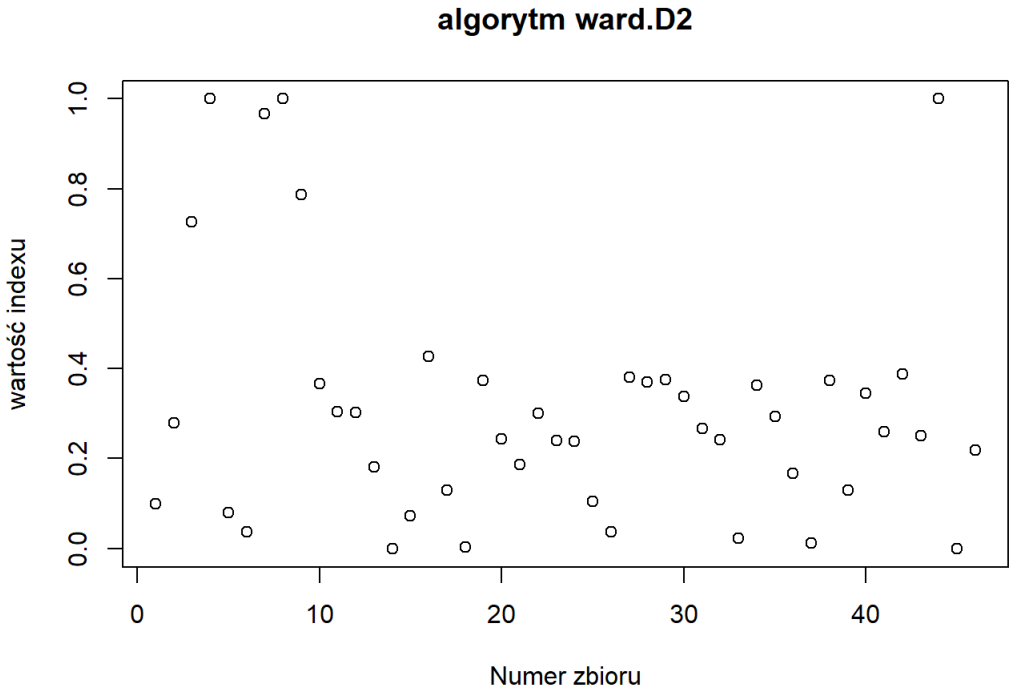


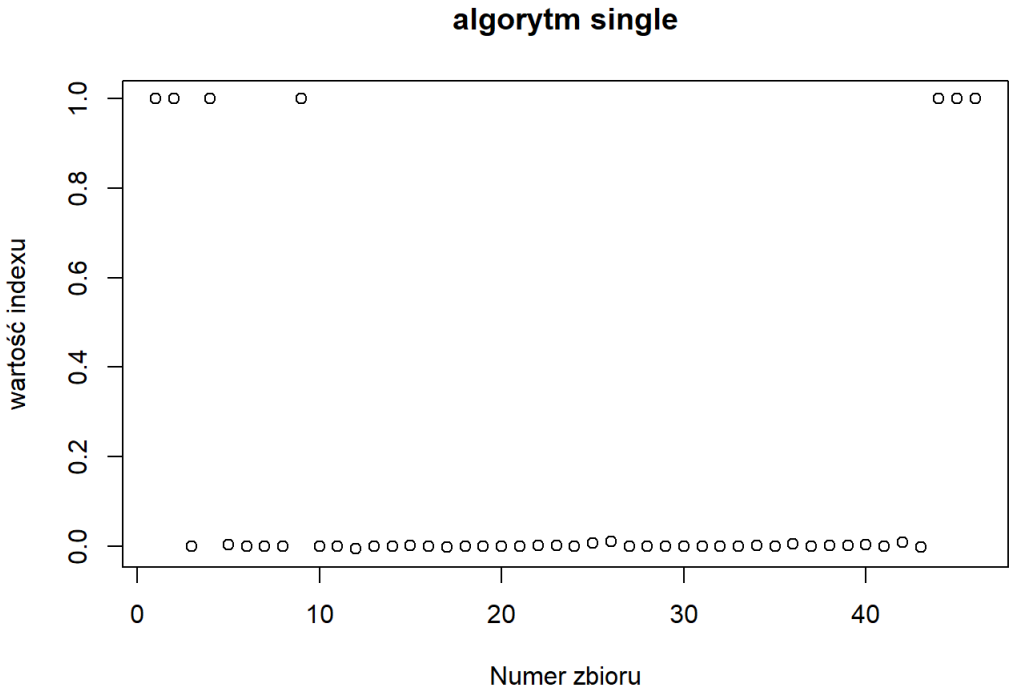


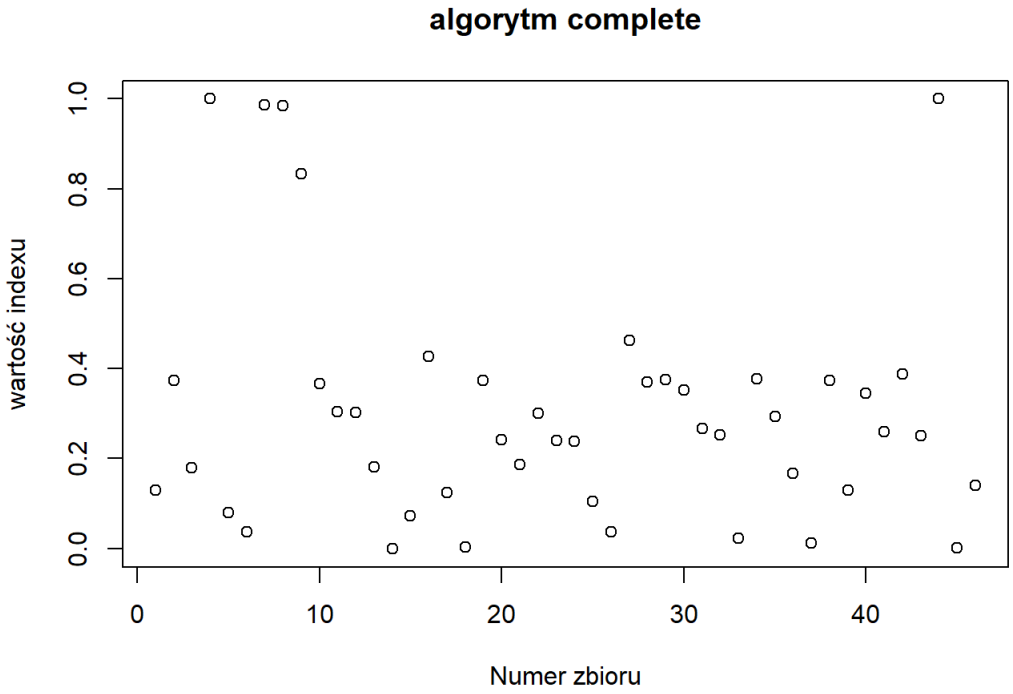
skorygowany indeks Rnada

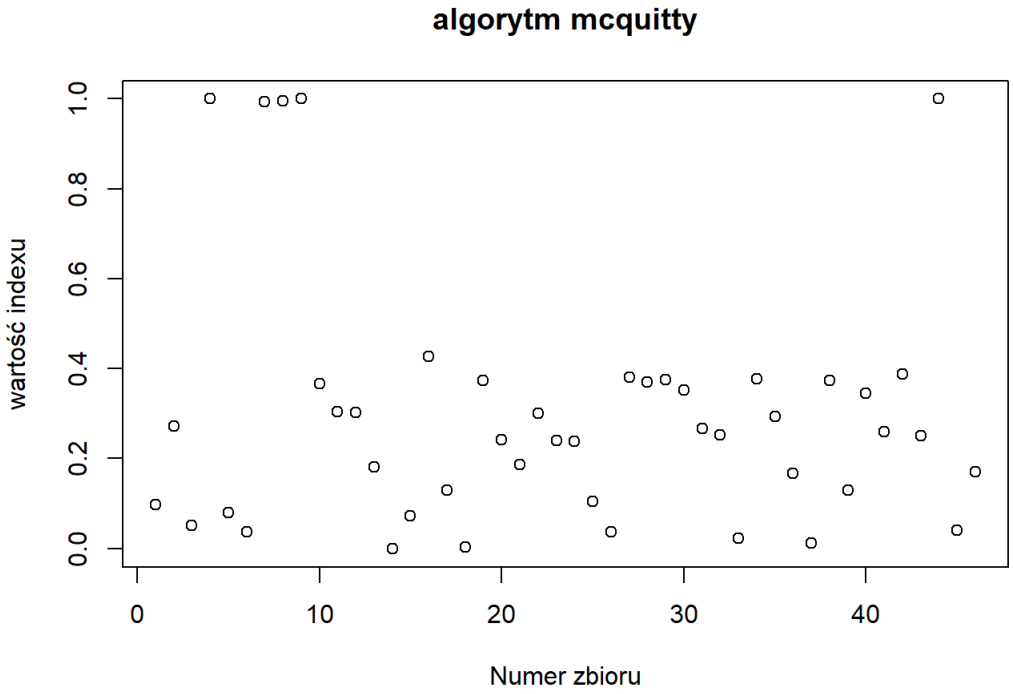

algorytm Spec_Clust
wartość indexu
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
Numer zbioru

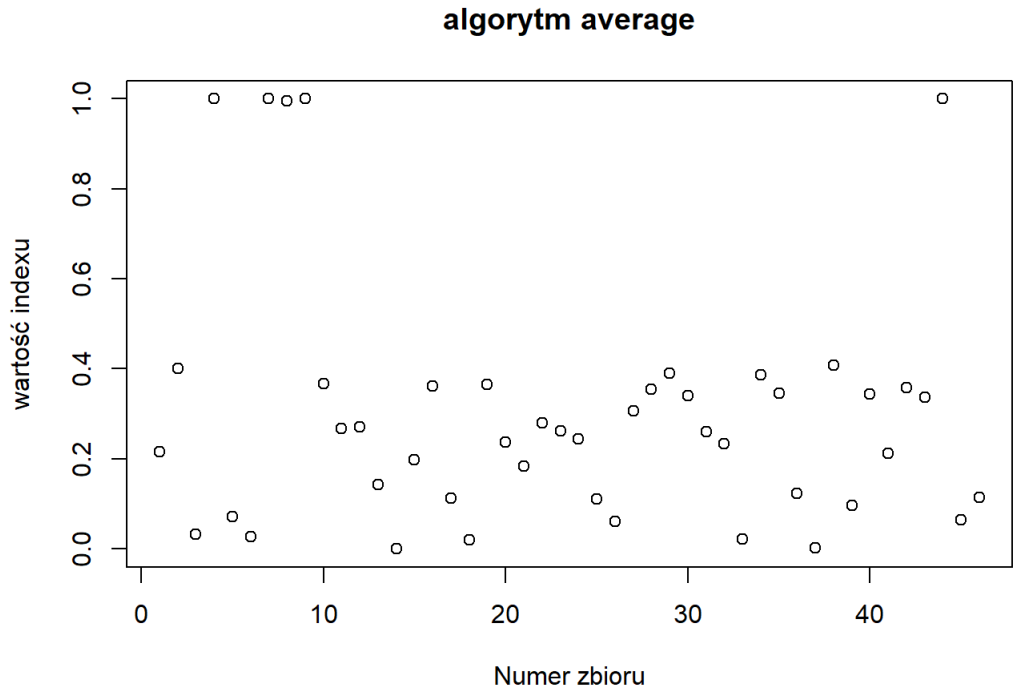

algorytm ward.D
wartość indexu
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

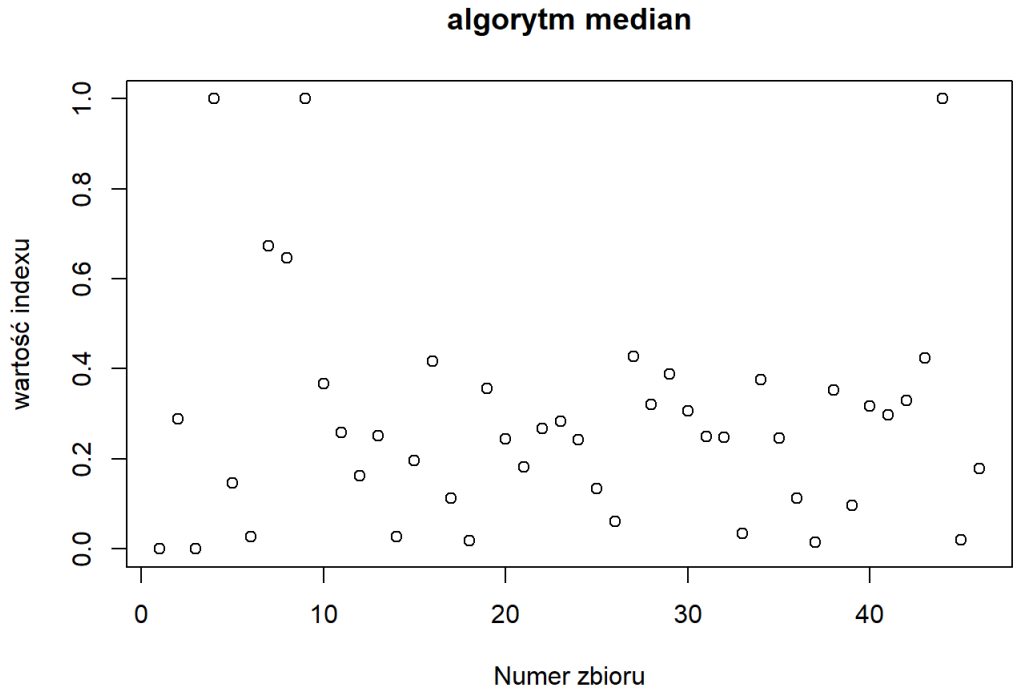

algorytm ward.D2
wartość indexu
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

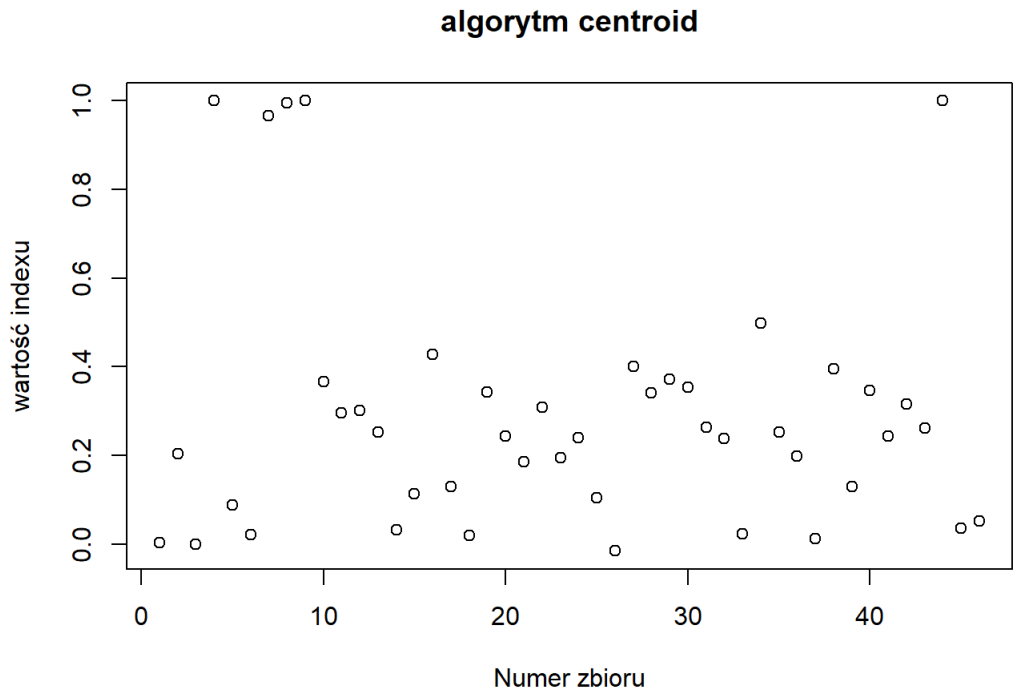

algorytm single
wartość indexu
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

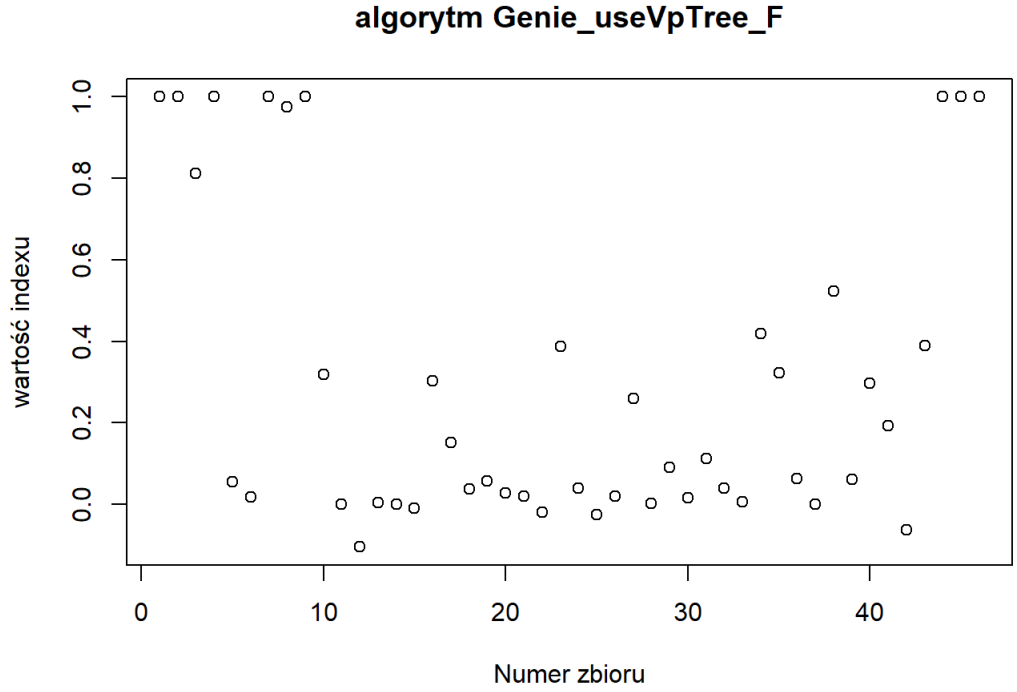

algorytm complete
wartość indexu
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru

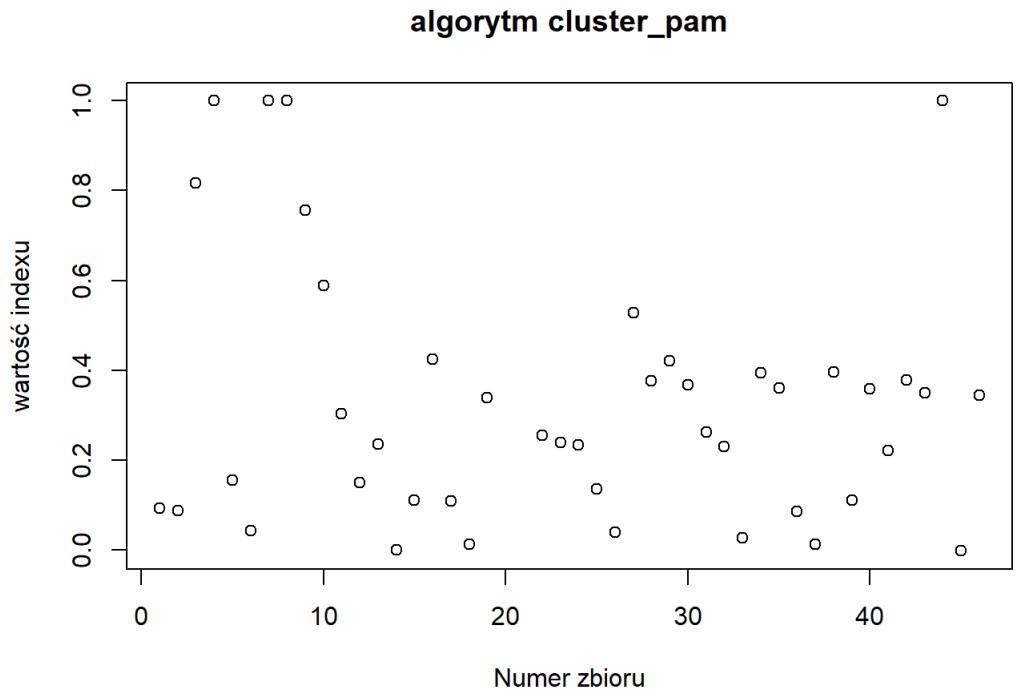

algorytm mcquitty
wartość indexu
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru


algorytm average
wartość indexu
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru


algorytm median
wartość indexu
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru


algorytm centroid
wartość indexu
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru


algorytm Genie_useVpTree_F
wartość indexu
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Numer zbioru



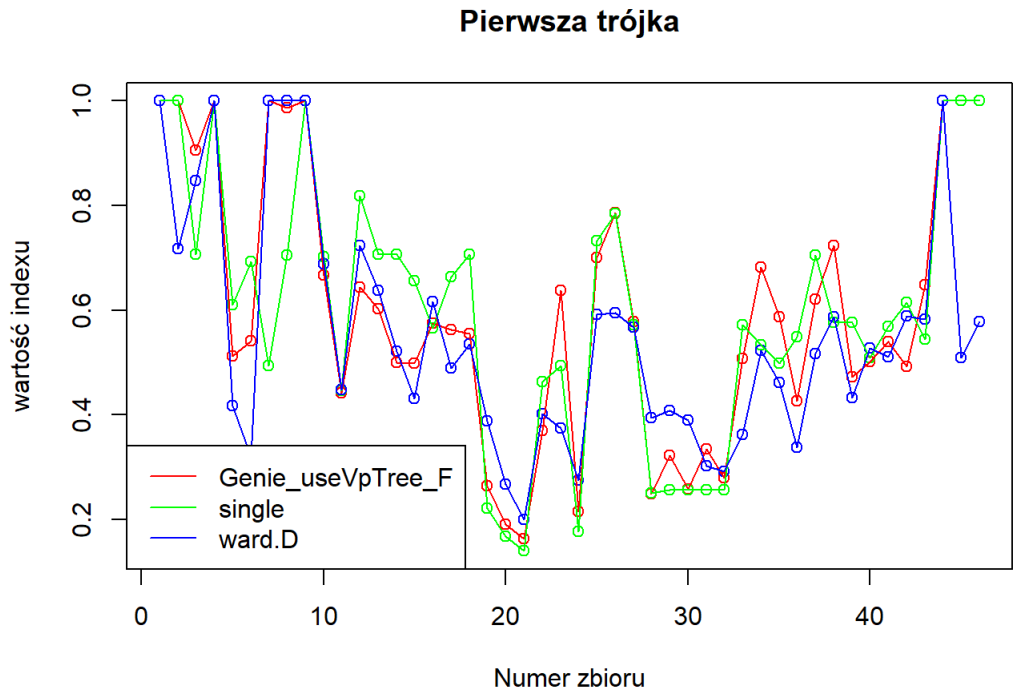
Nietrudno zauważyć, że takie zestawienie może i byłoby przydatne do dokładnej analizy jednak nie jest ono zbyt czytelne. Poniżej przedstawię zestawienia tych samych indeksów dla “najlepszych” trzech, i “najgorszych” trzech algorytmów. Jako kryterium, który algorytm jest lepszy posłużę się średnią wartością indeksów dla danych algorytmów.

indeksFowlkesa–Mallowsa

Srednie

```
##        Spec_Clust             ward.D            ward.D2             single 
##         0.4946398          0.5512346          0.5374724          0.5969519 
##           complete           mcquitty            average             median 
##         0.5348939          0.5388952          0.5359205          0.5271178 
##           centroid Genie_useVpTree_F        cluster_pam                    
##         0.5395144          0.5989187          0.5511302
```

Pierwsza trójka

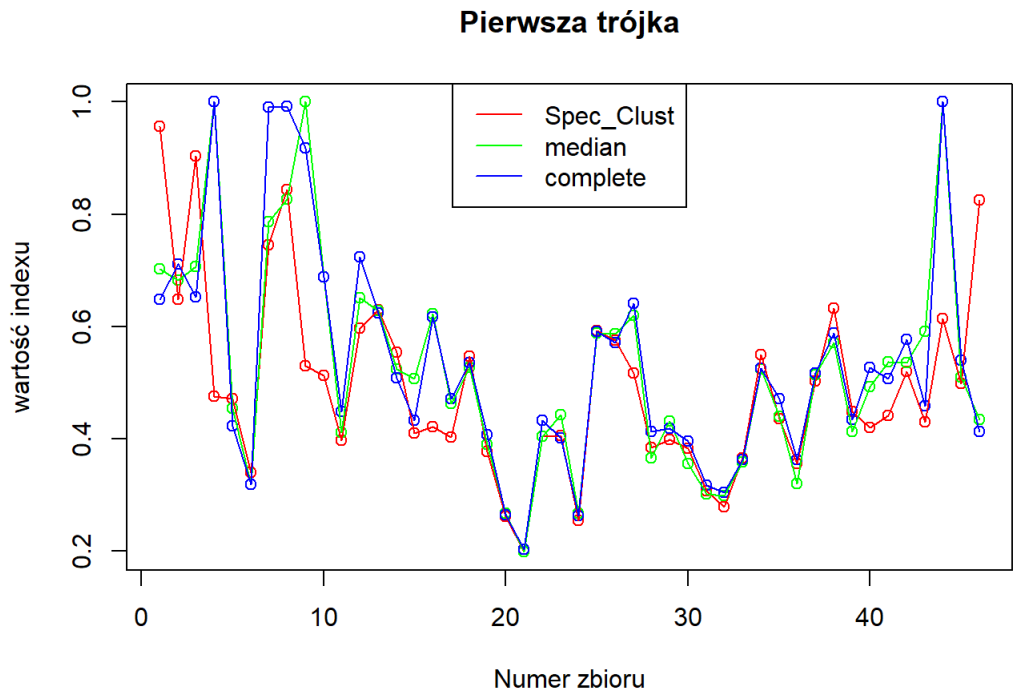


Prawidłowość zaobserwowana w średnich potwierdza się na wykresie. Algorytmy różnią się między sobą nieznacznie zwłaszcza pierwsze miejsce, które przypadło algorytmowi **Genie**. Zobaczmy jeszcze jak algorytmy wypadają w podstawowych parametrach rozkładu.

```
##                     średnia odchylenie standardowe   mediana   minimum
## Genie_useVpTree_F 0.5989187              0.2539891 0.5690529 0.1637386
## single            0.5969519              0.2460251 0.5762444 0.1400838
## ward.D            0.5512346              0.2192988 0.5188933 0.1994137
##                   maksimum
## Genie_useVpTree_F        1
## single                   1
## ward.D                   1
```

Możemy zaobserwować, że inne parametry istotne dla nas takie jak wartość minimalna, odchylenie standardowe, czy mediana są kożystniejsze dla algorytmu **single**, stąd i ze względu na minimanie gorszą średnią wyniku wnioskuję, że według indeksu **F-M**, metody **Genie** i **Single** są podobnie dobre.

Ostatnia trójka



Tutaj, podobnie jak przy analizie agorytmów najlepszych widzimy, że w gruncie rzeczy dane otrzymane z różnych algorytmów wypadają podobnie. według średniej najgorzej działa własnoręcznie zaimplementowana metoda analizy spektralnej. Może być to spowodowane *naiwnym* wyborem krawędzi uspójniających graf (więcej w testy.pdf).

```
##              średnia odchylenie standardowe   mediana   minimum  maksimum
## Spec_Clust 0.4946398              0.1662246 0.4597019 0.1981976 0.9558962
## median     0.5271178              0.1896949 0.5116981 0.1990625 1.0000000
## complete   0.5348939              0.1986051 0.5079674 0.2025136 1.0000000
```

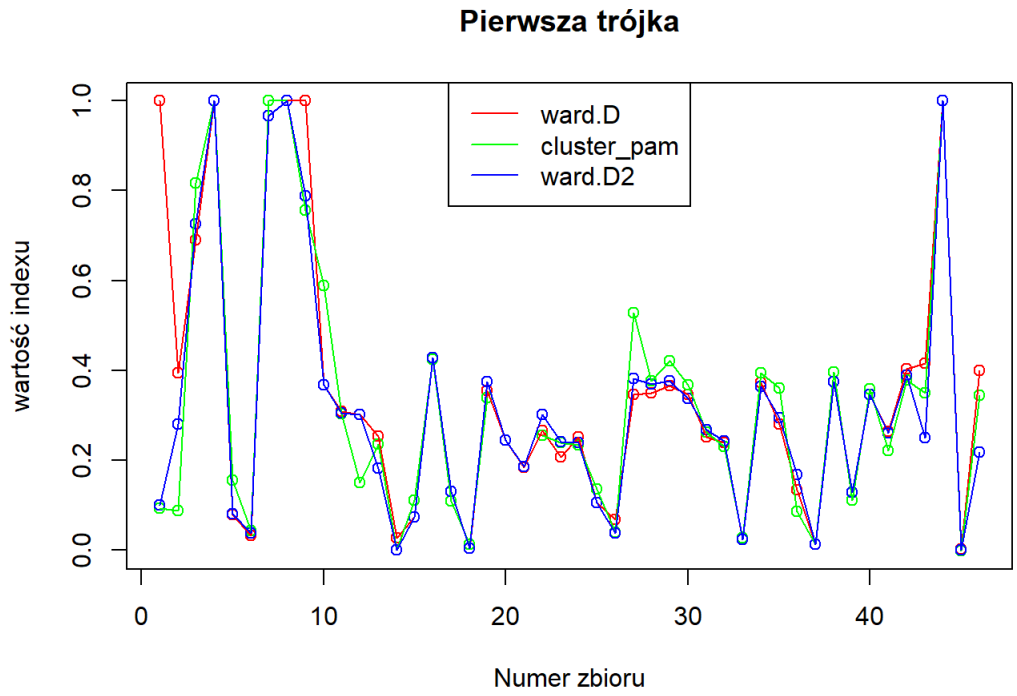
W tym wypadku widzimy, że niezależnie od doboru kryterium najgorzej wypada algorytm spektralny.

skorygowany Indeks Randa

Srednie

```
##        Spec_Clust            ward.D           ward.D2             single
##         0.2433178         0.3437393         0.3105902         0.1530643
##          complete          mcquitty           average            median
##         0.3032830         0.3014490         0.3022318         0.2848137
##          centroid Genie_useVpTree_F       cluster_pam
##         0.2946525         0.2997803         0.3258411
```

Pierwsza trójka

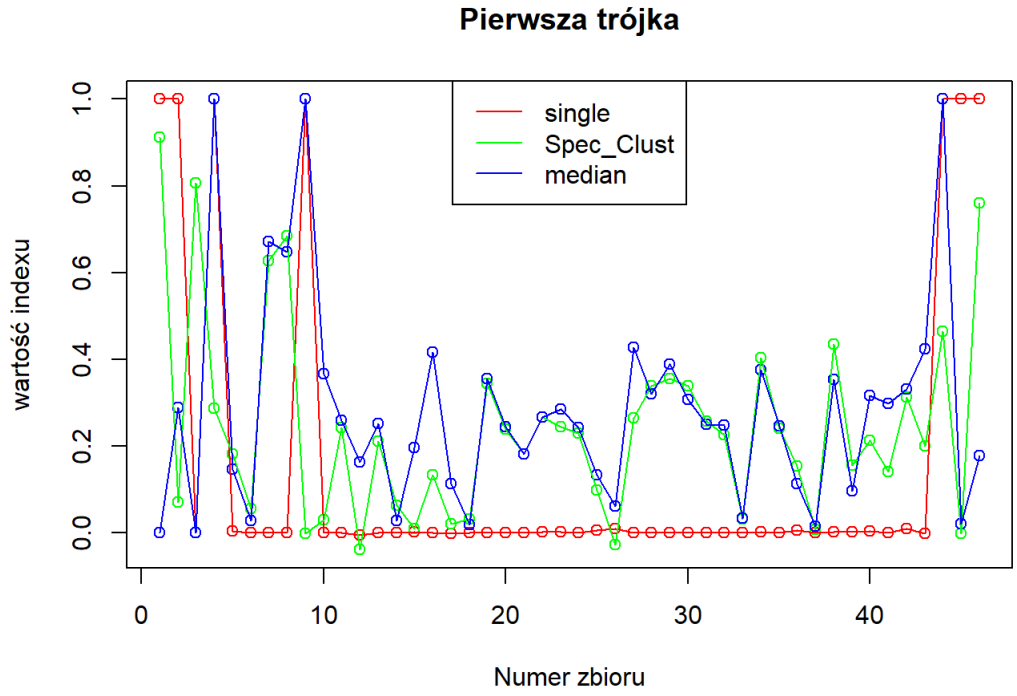


Mierząc drugim indeksem najlepszy okazał się algorytm *ward.D*. Co ciekawe dawał on niemal identyczne wyniki co pozostałe algorytmy "z podium". Jednak wynik dla jednego ze zbiorów miał znacząco lepszy. Zobaczmy jeszcze jak algorytmy wypadają w podstawowych parametrach rozkładu.

```
##              średnia odchylenie standardowe   mediana       minimum
## ward.D      0.3437393              0.2942266 0.2911736  0.0018184192
## cluster_pam 0.3258411              0.2860248 0.2583965 -0.0024874370
## ward.D2     0.3105902              0.2694827 0.2635854 -0.0009637907
##             maksimum
## ward.D             1
## cluster_pam        1
## ward.D2            1
```

W tym wypadku algorytm **ward.D** jest najlepszy w każdej klasyfikacji

Ostatnia trójka



Zdecydowanie najgorszym algorytmem wg. indeksu Randa jest algorytm **Single**. Co ciekawe przybiera on zwykle wartości indeku bardzo bliskie zeru. A większość *niezerowych* obserwacji jest obserwacjami bardzo bliskimi jedynki.

```
##              średnia odchylenie standardowe      mediana       minimum
## single     0.1530643              0.3627867 3.085533e-05 -5.961187e-03
## Spec_Clust 0.2433178              0.2247392 2.198208e-01 -3.907002e-02
## median     0.2848137              0.2461394 2.500801e-01 -2.384769e-07
##             maksimum
## single     1.0000000
## Spec_Clust 0.9118072
## median     1.0000000
```

Parametry rozkładu utwierdzają nas w przekonaniu, że według drugiego z indeksów algorytm single jest najgorszy.

Wnioski

- Analiza skupień pojęciem bardzo rozległym.
- Różne indeksy dały znacząco różne wyniki.
- Jeśli miałbym więcej czasu rozważyłbym jeszcze w analizie czas wykonywania algorytmów.
- Interfejs zapisu i odczytu plików w języku R nie należy do moich ulubionych.
- Napisanie funkcji w Rcpp było dużo przyjemniejsze niż myślałem, że będzie.

P.S.

- Informuję, że plik “benchmark_data.R” nie jest w stanie wygenerować kompletnych danych wykorzystanych w analizie. Stanowi jedynie szkielet takiego pliku.
- Zdaję sobie sprawę, że jakość kodu momentami nie jest najwyższa.
- W razie trudości w czytaniu pdf’a dołączam plik html’owy

Powyższe mankamenty jak i brak analizy czasu wykonania spowodowane były brakiem czasu.