Analiza ułożeń Kostki Rubika bez patrzenia w roku 2021 - Krzysztof Bober

Kostka Rubika dzieli się na 2 układalne typy elementów:

- krawędzie, elementy dwukolorowe, jest ich 12
- rogi, elementy trójkolorowe, jest ich 8.

Od kilku lat regularnie trenuję układanie Kostki Rubika bez patrzenia.Taka próba dzieli się na dwa procesy: zapamiętywanie i układanie, gdzie czas obejmuje oba te etapy (są one wobec siebie niezależne).
W 2021 roku jak dotąd udało mi się wykonać 803 udane ułożenia treningowe, które zawierają się w pliku “*bld_solves.csv*”.

```r
library(data.table)
library(moments)
solves = fread('3bld_solves.csv')
solves
```

```
> solves
        no result                                                          scramble                time algs
  1: 27435  23.70            F' R2 F R2 L2 B' L' U' F2 R' F2 R' L2 D2 F2 R B2 R' Rw2 2021-02-15 11:48:25   11
  2: 27437  22.63    D R2 B R U' B2 U F2 D' R2 D2 L2 B2 U2 R D L2 F2 L2 B' D' Fw Uw2 2021-02-15 11:49:36   11
  3: 27439  23.90     F' B D2 R2 U L B2 R U F2 R2 U2 B2 R2 F' D2 B' R2 D2 B U2 Fw Uw 2021-02-15 11:51:22   11
  4: 27442  20.30 U' R' B2 L2 D2 R' U2 F2 R' B2 L B2 D2 F R' B' F2 D' R B F' Rw2 Uw2 2021-02-15 11:53:41    9
  5: 27443  23.37               D2 F2 U R2 D' B2 D' B2 R2 U F' R2 F R U2 B2 R2 D Rw2 2021-02-15 11:54:11   10
 ---
799: 30169  19.42                  U F L D2 R U D' F R2 U F2 R2 F2 D B2 U' R2 U L2 U' 2021-02-12 20:37:44   10
800: 30171  22.13      U F U2 L2 F L2 U2 F2 L2 R2 U2 D F' L' R' F2 D' L2 D' U Fw Uw' 2021-02-12 20:39:18    9
801: 30172  20.49        D' F' R U' B' D R' L' B' U2 D2 L2 U L2 B2 U L2 B2 R2 U' Rw' 2021-02-12 20:40:03   10
802: 30179  19.29          B2 L2 U2 B2 U' B2 U' F2 D' L2 R2 B F D' B' R D L' D B2 D' 2021-02-12 20:47:43    9
803: 30182  23.73                B U B' D2 B' R2 B L2 F' R' D R F' U' R' D2 L Rw Uw' 2021-02-12 20:50:15   10
     edges corners flips twists solved_edges solved_corners edge_breaks corner_breaks
  1:    13       7     0      0            0              0           2             0
  2:    11       7     0      1            1              0           1             1
  3:    11       9     0      0            0              0           0             2
  4:    10       6     0      1            2              0           1             0
  5:    12       6     0      1            1              0           2             0
 ---
799:    12       8     0      0            0              1           1             2
800:    12       6     0      0            0              2           1             1
801:    11       7     0      0            0              0           0             0
802:    10       6     1      0            0              1           0             0
803:     9       9     0      0            2              0           0             2
```

Ramka danych *solves* składa się z 803 wierszy i 13 kolumn:

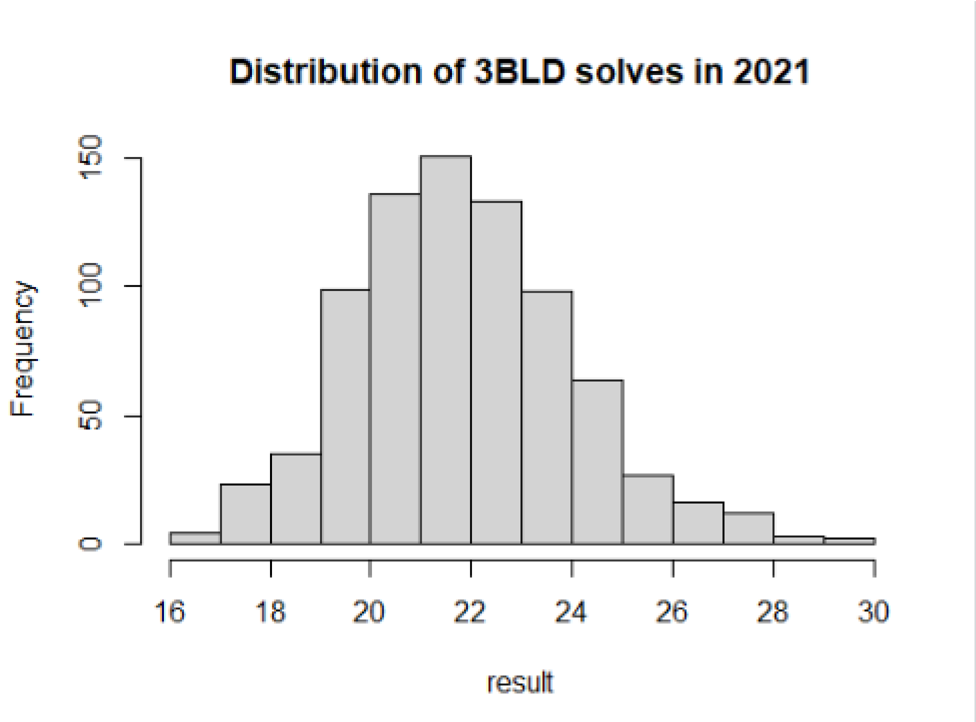
- no - numer próby ułożenia (od rozpoczęcia archiwizowania danych przeze mnie, tj. 21 maja 2019 r.)
- result - czas próby (w sekundach)
- scramble - algorytm mieszający Kostkę Rubika
- time - czas wykonania próby
- algs - liczba algorytmów potrzebnych do ułożenia Kostki
- edges, corners - liczba pomieszanych krawędzi i rogów
- flips, twists - liczba odwróconych wokół własnej osi kolejno krawędzi i rogów
- solved_edges, solved_corners - liczba ułożonych krawędzi i rogów
- edge_breaks, corners_breaks - liczba włamań do cyklu kolejno krawędzi i rogów.

W niektórych ułożeniach liczba pomieszanych krawędzi i rogów jest większa od ich liczby na kostce (czyli 12 i 8). Jest to spowodowane włamaniami do cyklu, co oznacza, że te wartości są od siebie bardzo zależne.

Oczywiście część danych jest zupełnie zbędna w dalszych działaniach.

```
> solves <- solves[,c(2,5,6,7,8,9,10,11,12,13)]
> solves
     result algs edges corners flips twists solved_edges solved_corners edge_breaks corner_breaks
  1:  23.70   11    13       7     0      0            0              0           2             0
  2:  22.63   11    11       7     0      1            1              0           1             1
  3:  23.90   11    11       9     0      0            0              0           0             2
  4:  20.30    9    10       6     0      1            2              0           1             0
  5:  23.37   10    12       6     0      1            1              0           2             0
 ---
799:  19.42   10    12       8     0      0            0              1           1             2
800:  22.13    9    12       6     0      0            0              2           1             1
801:  20.49   10    11       7     0      0            0              0           0             0
802:  19.29    9    10       6     1      0            0              1           0             0
803:  23.73   10     9       9     0      0            2              0           0             2
```

```
> summary(solves$result)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  16.12   20.32   21.71   21.85   23.20   29.56 
```



Łatwość stanu pomieszania Kostki Rubika (dalej nazywany jako scramble) głównie zależy od liczby algorytmów potrzebnych do jej ułożenia.

```
> summary(solves$algs)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   7.00   10.00   10.00   10.46   11.00   13.00 
> table(solves$algs)

  7   8   9  10  11  12  13 
  1  12 103 299 289  90   9 
```

Czynnikiem, która znacząco wpływa na czas próby jest jakość zapamiętania układanki. Bardzo często można zapomnieć, który element powinien zostać ułożony jako kolejny co powoduje wzrost czasu próby. Wówczas oczywiście taka próba jest odstająca.
Rezultaty hipotetycznie odstające od najlepszych wyników (jako jeszcze lepsze) nie zostaną poddane badaniu czy są odstające, ponieważ żaden czynnik nie wpływa mocniej na łatwość scrambla niż na jego trudność.
Na samym początku podzielmy dane na 7 klas, które zależne są od liczby algorytmów potrzebnych do ułożenia kostki.

```
seven_algs <- solves[solves$algs==7]
eight_algs <- solves[solves$algs==8]
nine_algs <-   <- solves[solves$algs==9]
ten_algs <- solves[solves$algs==10]
eleven_algs <- solves[solves$algs==11]
twelve_algs <- solves[solves$algs==12]
thirteen_algs <- solves[solves$algs==13]
```

Dla każdej klasy sprawdźmy wartości odstające używając testu Grubbsa. Aby uniknąć redundancji stworzona zostaje funkcja *remove_outlier()*, która zwraca ramkę danych bez elementu odstającego.

```
remove_outlier <- function(data){
  grubbs.test(data$result)
  if(grubbs.test(data$result)$p.value <= 0.05)
  {
    res <- data[order(data$result)]
    return(res[-c(2)])
  }
  else{
    return(data)
  }


  return(data)
}
```

Funkcja jest użyta dla każdej klasy, a ich zwartości jest połączona ponownie w jedną ramkę.

```

solves_wo_outliers <- rbind(seven_algs, remove_outlier(eight_algs),
                            remove_outlier(eleven_algs), remove_outlier(nine_algs),
                            remove_outlier(ten_algs), remove_outlier(twelve_algs),
                            remove_outlier(thirteen_algs))


solves <- solves_wo_outliers[order(solves_wo_outliers$result)]
```

```
> summary(solves$result)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  16.12   20.35   21.73   21.86   23.20   29.56 
> 
```

```
> dim(solves)
[1] 800  10
```

Usunięto 3 odstające rezultaty.

Następnie używając testu Shapiro-Wilka sprawdzamy czy rozkład czasów bez danych odstających jest zbliżony do rozkładu normalnego.

```
> solves.normality <- shapiro.test(solves$result)
> solves.normality

	Shapiro-Wilk normality test

data:  solves$result
W = 0.99101, p-value = 8.567e-05

> 
```

Wartość współczynnika p jest bardzo niska, więc rozkład czasów jest zbliżony do rozkładu normalnego.

Również sprawdzamy kurtozę i skośność tych danych:

```
> skewness(solves$result)
[1] 0.3692707
> kurtosis(solves$result)
[1] 3.243144
> 
```

Aby lepiej przygotować dane do pracy z nimi poddajmy je przekształceniu. W tym wypadku użyte zostanie przekształcenia Tukeya.

```
library(rcompanion)

transformed_solves <- apply(solves, 2, function(x) transformTukey(x, quiet = TRUE))
head(transformed_solves)
```

```
> head(transformed_solves)
         result     algs    edges  corners flips twists solved_edges solved_corners edge_breaks corner_breaks
[1,] -0.4343007 8.426888 74.98942 15.37024     0      0      1.71119              1           1             0
[2,] -0.4311188 8.426888 74.98942 15.37024     0      0      1.00000              1           0             0
[3,] -0.4309624 8.426888 74.98942 15.37024     0      0      1.00000              1           0             0
[4,] -0.4284920 9.508206 74.98942 23.83484     0      0      1.00000              0           0             1
[5,] -0.4269788 9.508206 74.98942 23.83484     0      0      1.71119              0           1             1
[6,] -0.4267538 9.508206 74.98942 15.37024     1      0      1.00000              1           1             0
> 
```

Przejdźmy teraz do analizy głównych składowych używając metody PCA. Argumenty funkcji “scale” i “center” zostaną ustawione jako TRUE, aby odpowiednio skoncentrować

dane wokół zera i przeskalować je tak, aby różnice w wielkościach danych zostały zmarginalizowane.
Od razu sprawdźmy wyniki metody PCA dla danych przed i po transformacją Tukeya.

```r

solves.PCA <- prcomp(solves, center = TRUE, scale=TRUE)
transformed_solves.PCA <- prcomp(transformed_solves, center = TRUE, scale=TRUE)

```

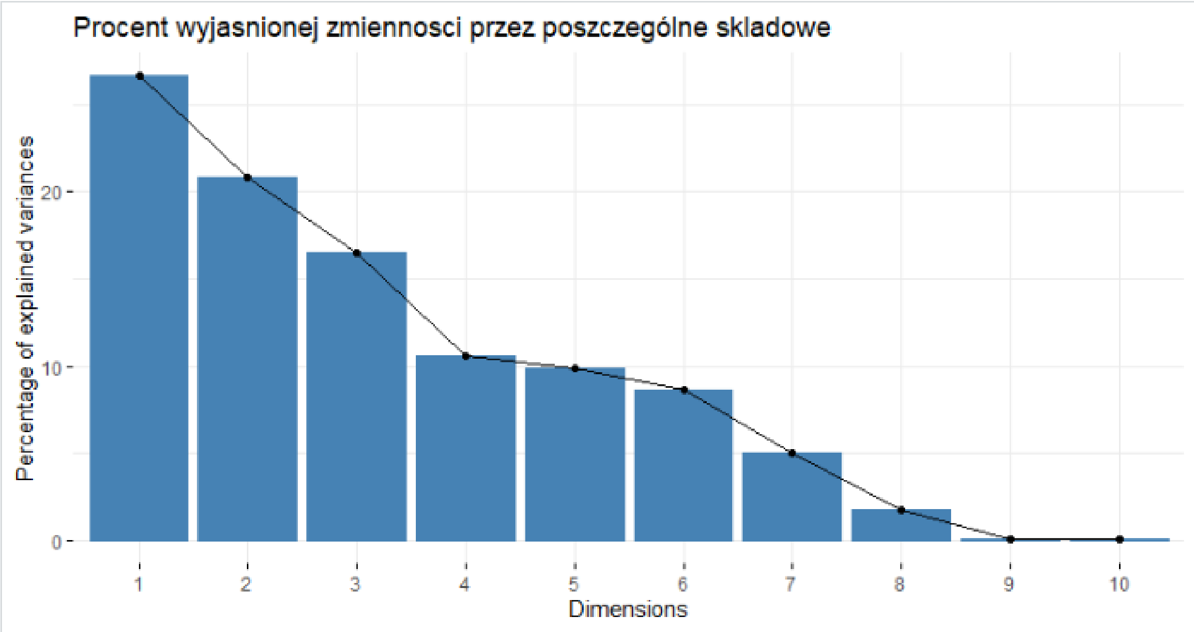
Spójrzmy na wyniki:

```
> solves.PCA
Standard deviations (1, .., p=10):
 [1] 1.625620e+00 1.444902e+00 1.295066e+00 1.024887e+00 9.942732e-01 9.342902e-01 7.177273e-01 4.067210e-01 9.400189e-16 7.630877e-16

Rotation (n x k) = (10 x 10):
                      PC1          PC2         PC3         PC4          PC5         PC6          PC7         PC8           PC9          PC10
result         0.38566045 -0.123106816  0.35119373 -0.02967288  0.050423713  0.06990031  0.835658796 -0.07835168  7.923939e-17  1.888297e-16
algs           0.49976272 -0.113233614  0.31906689  0.01666271  0.002121888  0.05228565 -0.316927900  0.72948781  2.305040e-16 -4.117272e-16
edges          0.43307147 -0.197056438 -0.49947889 -0.01578239 -0.024027224 -0.04086158 -0.025606753 -0.11658080  3.918524e-01 -5.948508e-01
corners        0.23731898  0.637217861 -0.03093972  0.01202240 -0.029664677 -0.02125755 -0.003366141 -0.05026759  6.094810e-01  4.014899e-01
flips         -0.05310667  0.001015761  0.60566627 -0.21206527 -0.536402972 -0.01573334 -0.233962044 -0.32248244  2.048495e-01 -3.109713e-01
twists         0.02371548 -0.489962599  0.26375850  0.24207162  0.508519645  0.11225184 -0.256792302 -0.33428297  3.565825e-01  2.348954e-01
solved_edges  -0.32091928  0.140885288  0.08015450  0.65608263 -0.089605449  0.47466085  0.144882644  0.22086622  2.037545e-01 -3.093092e-01
solved_corners -0.22158907 -0.256507625 -0.16663767 -0.57139079 -0.113376408  0.60588892  0.088500823  0.19328056  2.715030e-01  1.788500e-01
edge_breaks    0.36481244 -0.188601628 -0.21599916  0.32819354 -0.537089411  0.30252358 -0.111047847 -0.26059144 -2.561110e-01  3.887888e-01
corner_breaks  0.26281225  0.407265562  0.08470057 -0.17388229  0.374602304  0.54174326 -0.196795634 -0.27532427 -3.536207e-01 -2.329443e-01
> summary(solves.PCA)
Importance of components:
                          PC1    PC2    PC3    PC4     PC5     PC6     PC7     PC8     PC9      PC10
Standard deviation     1.6256 1.4449 1.2951 1.0249 0.99427 0.93429 0.71773 0.40672 9.4e-16 7.631e-16
Proportion of Variance 0.2643 0.2088 0.1677 0.1050 0.09886 0.08729 0.05151 0.01654 0.0e+00 0.000e+00
Cumulative Proportion  0.2643 0.4730 0.6408 0.7458 0.84465 0.93194 0.98346 1.00000 1.0e+00 1.000e+00

> transformed_solves.PCA
Standard deviations (1, .., p=10):
 [1] 1.63208727 1.44451571 1.28610178 1.02903599 0.99273208 0.92955064 0.70884068 0.41742358 0.07746030 0.06643501

Rotation (n x k) = (10 x 10):
                      PC1           PC2         PC3         PC4          PC5         PC6         PC7         PC8          PC9         PC10
result         0.39677153  0.1164336684 -0.34245027 -0.02345317  0.057543627 -0.06066149 -0.83469373 -0.08647221  0.002963776  0.001543635
algs           0.50124011  0.1075056901 -0.31217119  0.02313432  0.001538121 -0.05686747  0.30873552  0.73528704 -0.003538909 -0.001176522
edges          0.42799942  0.2030463755  0.50234606 -0.02183854 -0.023330264  0.04188402  0.03149819 -0.12096413 -0.704716729 -0.090617238
corners        0.23610923 -0.6362309530  0.03964400  0.01304847 -0.030100300  0.01757413  0.01006213 -0.05360441 -0.093520192  0.724420536
flips         -0.03780974 -0.0006474266 -0.60878614 -0.21916576 -0.538928919  0.01604790  0.23157417 -0.32237248 -0.359761016 -0.047278860
twists         0.02484282  0.4814586990 -0.26871348  0.27203853  0.497952912 -0.14531047  0.26137981 -0.33160880 -0.055845488  0.413392917
solved_edges  -0.31744748 -0.1379427579 -0.05984098  0.64456289 -0.144896903 -0.48445183 -0.16265606  0.22021645 -0.358072005 -0.047959358
solved_corners -0.21715147  0.2502796428  0.15251853 -0.59713088 -0.086012804 -0.59804817 -0.09576582  0.18941304 -0.042387896  0.313804392
edge_breaks    0.36550489  0.1971036647  0.23583276  0.28439039 -0.544790208 -0.29395382  0.10815634 -0.25140886  0.478635198  0.062130956
corner_breaks  0.26235739 -0.4201752189 -0.07788655 -0.15272352  0.362828978 -0.53933889  0.19391620 -0.27000438  0.054372270 -0.435013897
> summary(transformed_solves.PCA)
Importance of components:
                          PC1    PC2    PC3    PC4     PC5     PC6     PC7     PC8     PC9    PC10
Standard deviation     1.6321 1.4445 1.2861 1.0290 0.99273 0.92955 0.70884 0.41742 0.07746 0.06644
Proportion of Variance 0.2664 0.2087 0.1654 0.1059 0.09855 0.08641 0.05025 0.01742 0.00060 0.00044
Cumulative Proportion  0.2664 0.4750 0.6404 0.7463 0.84488 0.93129 0.98153 0.99896 0.99956 1.00000
```

Jak widać zarówno dane poddane transformacji jak i te niestransformowane dają podobne rezultaty w macierzach, więc również w ważności składowych głównych.
W obu przypadkach pierwsza składowa wyjaśnia ~26.6% zmienności, a druga około 6 punktów procentowych mniej. Oznacza to, że rzutowanie na przestrzeń dwuwymiarową pozwoli nam na wyjaśnienie mniej niż 50% zmienności.

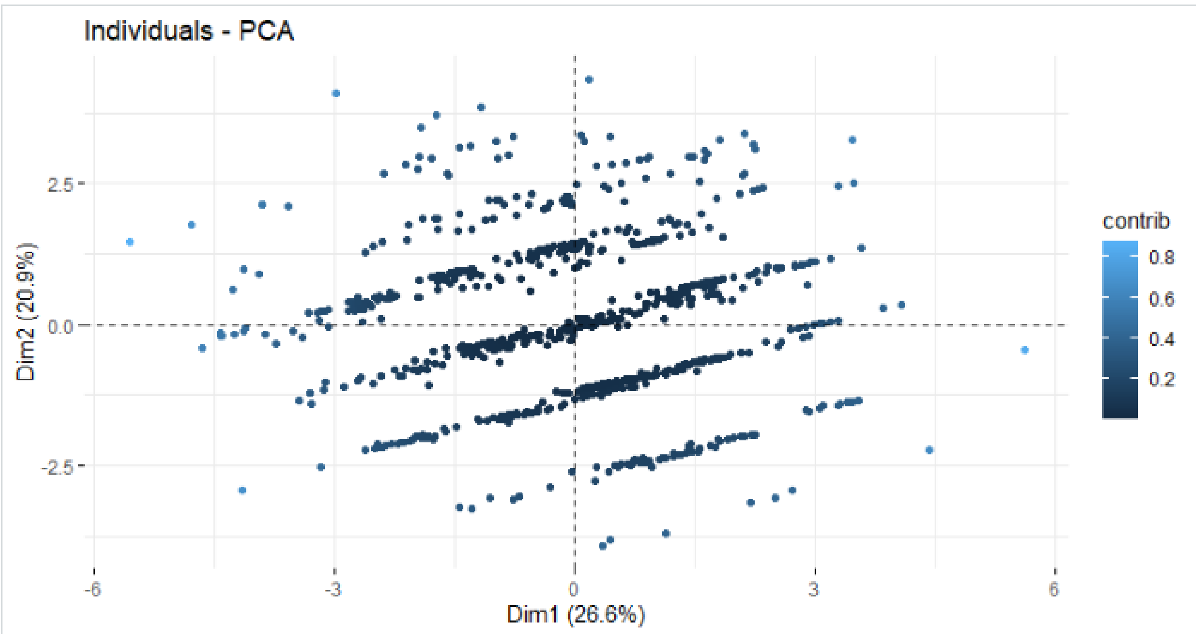
Spróbujmy przedstawić te dane na wykresie osypiskowym:

```r
library(ggplot2)
library(factoextra)
fviz_eig(transformed_solves.PCA, main="Procent wyjasnionej zmiennosci przez poszczególne skladowe")

```

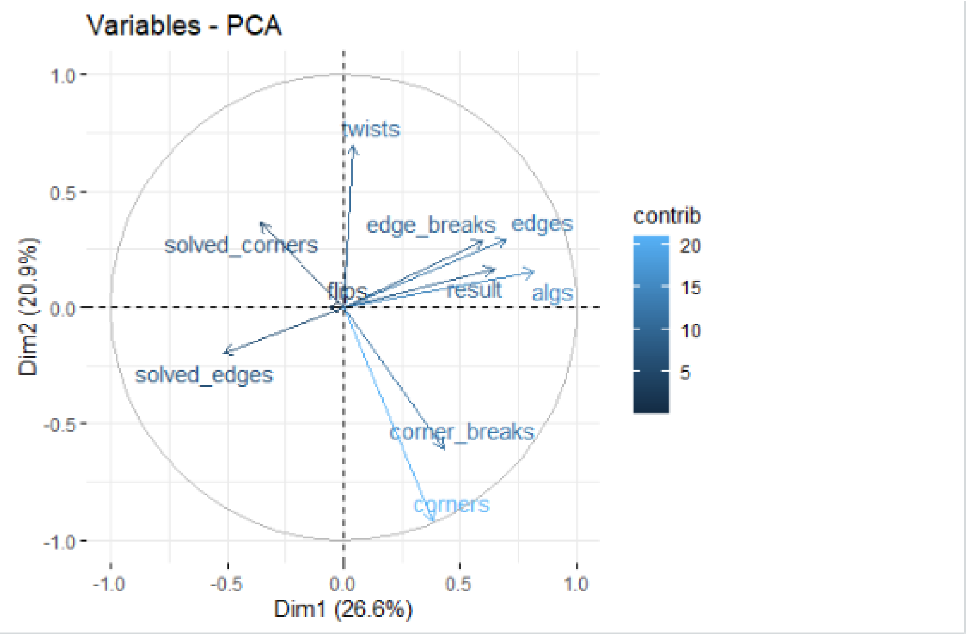


Teraz używając biblioteki *ggbiplot* sprawdźmy wartość dwóch głównych składowych dla poszczególnych obserwacji, a także korelację pomiędzy głównymi składowymi.

```
library(devtools)
install_github("vqv/ggbiplot")


fviz_pca_ind(transformed_solves.PCA, col.ind="contrib", label="none")
fviz_pca_var(transformed_solves.PCA, col.var="contrib", repel=TRUE )
```



Ewidentnie widać tutaj 7 grup danych, tyle samo ile było klas danych ze względu na liczbę algorytmów, aczkolwiek trzeba pamiętać, że PCA nie jest metodą klasteryzacji.



Natomiast ten wykres może posłużyć jako podstawa kilku trywialnych wniosków:

- pozytywnie skorelowane są takie zmienne jak liczba algorytmów i czas próby, lub liczba krawędzi/rogów i liczba włamań do cyklu krawędzi/rogów
- negatywnie skorelowane są zmienne wpływające na czas próby do liczby ułożonych krawędzi.

Z tych pozornie trywialnych zdań można wywnioskować, że liczba pomieszanych krawędzi jest o wiele ważniejsza niż liczba ułożonych rogów (ważniejsza w kontekście szans na uzyskanie lepszego wyniku).

Również możemy wysnuć następne ciekawe wnioski, do których pewnie bym nie doszedł bez testu PCA :

- Bardzo niska korelacja pomiędzy zmiennymi opisującymi rogi (t*wists*, *corner_breaks*, *corners*), a czasem próby. Zwłaszcza w porównaniu do danych krawędzi. Poniekąd jest to oczywiste (rogów jest mniej niż krawędzi), aczkolwiek różnica korelacji pomiędzy czasami a krawędziami i czasami a rogami jest ogromna.
- Wektor *corners* jest dłuższy niż wektor *egdes* co pokazuje jego większą wariancję, a przecież rogów jest mniej.

Co ciekawe powyższe wnioski przekładają się na dane z mojego ułożenia w którym pobiłem rekord Polski w Kostce Rubika bez patrzenia. Co prawda było to w 2019 roku (prawie dwa lata przed pobieraną próbą do tej analizy) i było to raptem jedno ułożenie, ale w dużym skrócie:

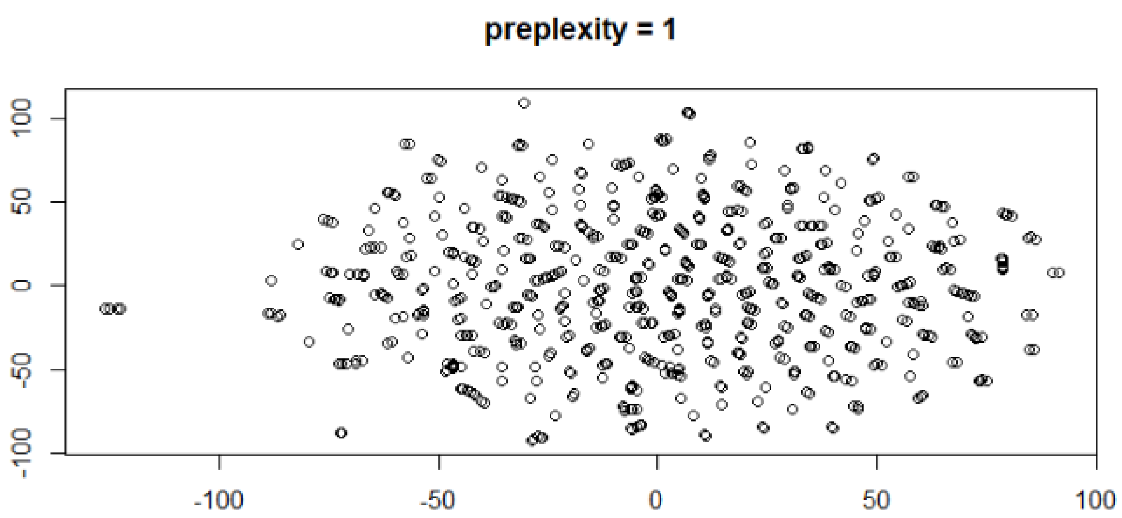
- liczba krawędzi: 10 - standardowo.
- liczba rogów: 8 - dużo

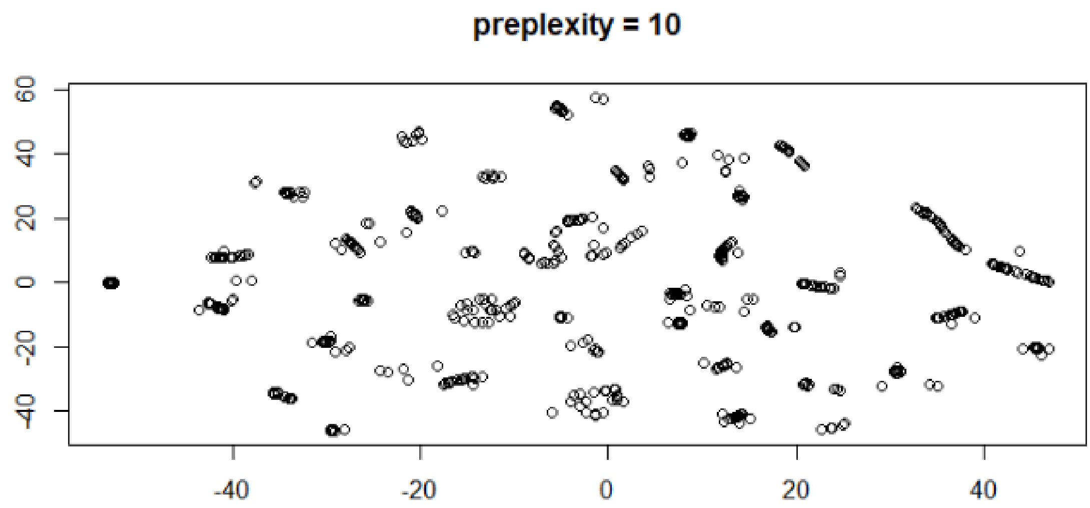
Wpływ na taką “wagę” krawędzi co do próby zapewne ma fakt, że algorytmy używane podczas układania tego elementu są średnio krótsze (tj. zawierają mniej ruchów).

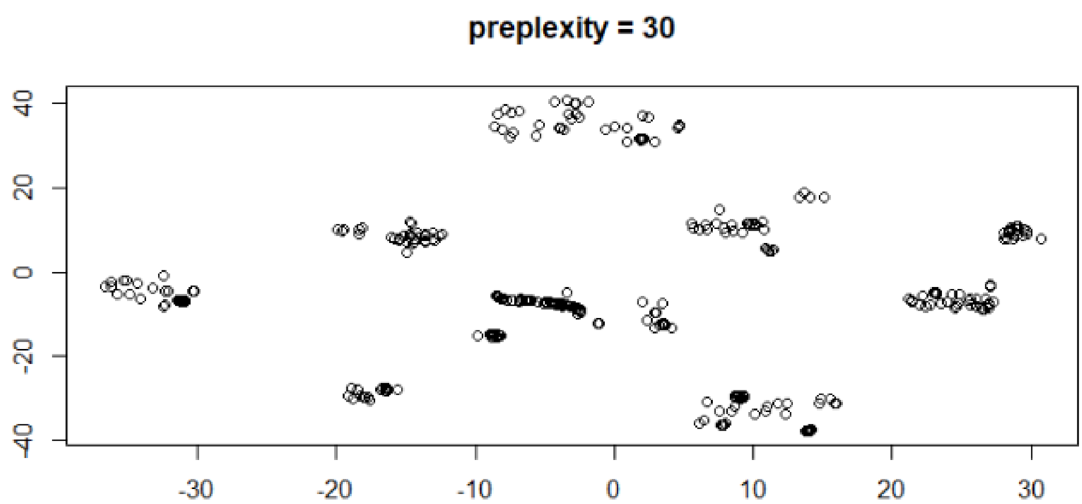
Oczywiście ważnym jest fakt, że korelacja nie oznacza zależności, jednakże porównując otrzymane wyniki z moim doświadczeniem w tej dziedzinie wyniki można uznać za na pewno słuszne.

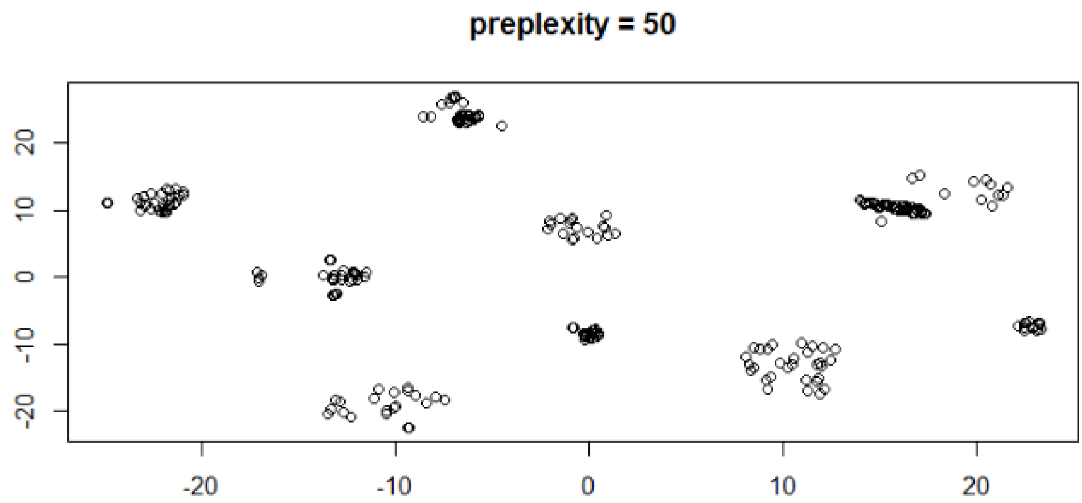
Używając metody t-SNE zbadajmy ponownie dane poddane trasnformacji za każdym razem jednak ustalając inna wartość dla argumenty *preplexity*.

```
library(Rtsne)

plot(Rtsne(transformed_solves, perplexity=1, check_duplicates = FALSE)$Y,
     main="preplexity = 1", xlab='', ylab='')
plot(Rtsne(transformed_solves, perplexity=10, check_duplicates = FALSE)$Y,
     main="preplexity = 10", xlab='', ylab='')
plot(Rtsne(transformed_solves, perplexity=30, check_duplicates = FALSE)$Y,
     main="preplexity = 30", xlab='', ylab='')
plot(Rtsne(transformed_solves, perplexity=50, check_duplicates = FALSE)$Y,
     main="preplexity = 50", xlab='', ylab='')
```




preplexity = 10
60
40
20
0
-20
-40
-40
-20
0
20
40

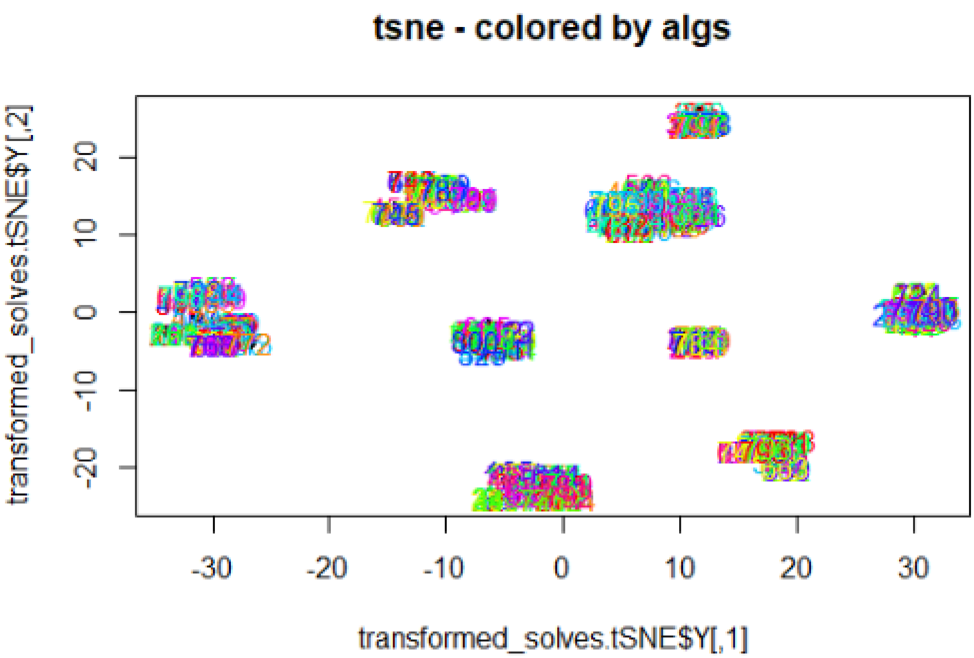

preplexity = 30
40
20
0
-20
-40
-30
-20
-10
0
10
20
30

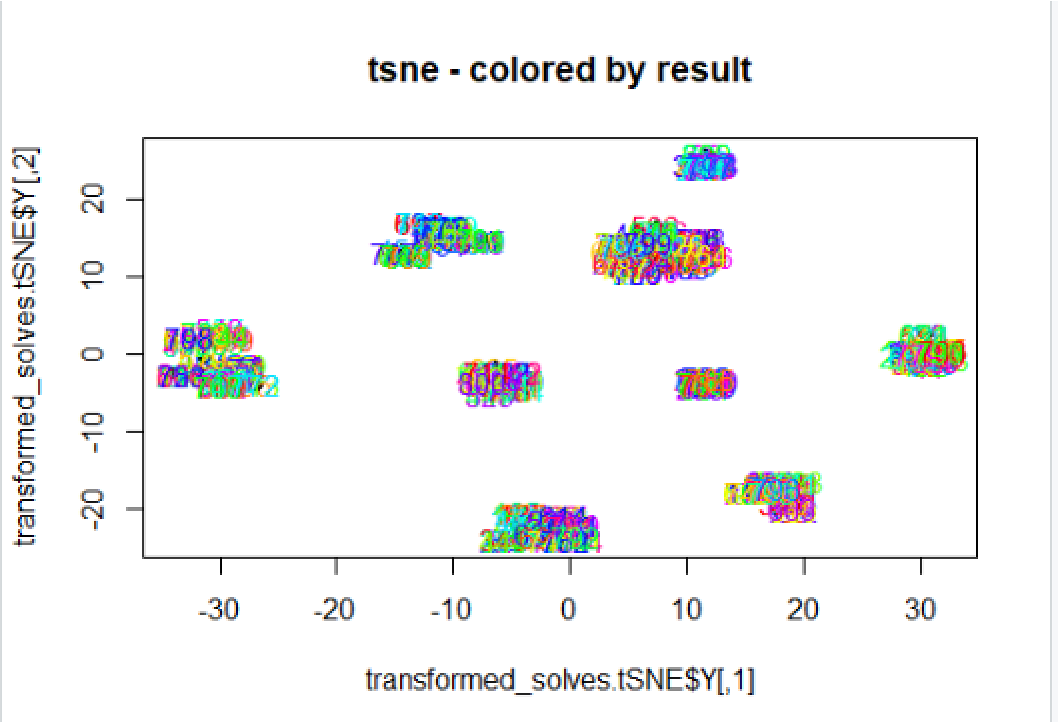


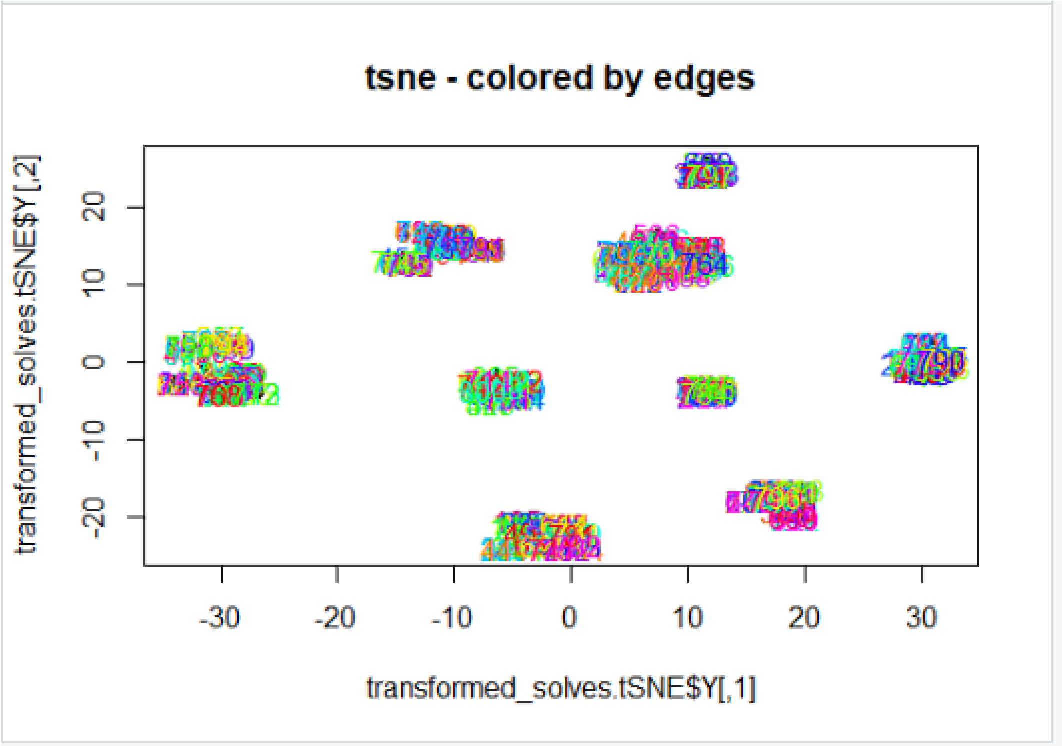
Wzrost wartości tego współczynnika powoduje coraz większy podział danych na grupy (chociaż tak samo jak PCA nie jest to metoda klasteryzacji).

Spróbujmy oznaczyć różnymi kolorami zmienne *algs*, *result*, *edge* w zależności od ich różnej wartości.

```
transformed_solves.tSNE <- Rtsne(transformed_solves, perplexity=50, check_duplicates = FALSE)

plot(transformed_solves.tSNE$Y, main = 'tsne - colored by algs', pch=16, label=NULL)
text(transformed_solves.tSNE$Y, col = rainbow(solves$algs))
plot(transformed_solves.tSNE$Y, main = 'tsne - colored by result', pch=16, label=NULL)
text(transformed_solves.tSNE$Y, col = rainbow(solves$result))
plot(transformed_solves.tSNE$Y, main = 'tsne - colored by edges', pch=16, label=NULL)
text(transformed_solves.tSNE$Y, col = rainbow(solves$edges))
```







Na żadnym z wykresów nie można zobaczyć żadnych wyraźnych jednokolorowych grup, ani nawet koloru dominującego wśród jakiejkolwiek grupy. Były to zmienne, które najbardziej “podejrzewałem” o największy wpływ na pozostałe biorąc pod uwagę wyniki testu PCA. Wykresy zostały stworzone przy współczynniku preplexity równym 50. Jego zmiana nie wpływała na zmianę.

Wnioski końcowe:

- Na czas próby ułożenia Kostki wpływa wiele czynników jednak najważniejsze z nich oprócz łącznej liczby algorytmów są liczba pomieszanych i ułożonych krawędzi
- Flipy (krawędzie na swoim miejscu lecz odwrócone) mają bardzo mały wpływ na czas próby, pomimo, że jest to element dotyczący krawędzi
- Obroty rogów (twisty) są mniej skorelowane z pomieszanymi rogami niż obroty krawędzi (flipy) z liczbą pomieszanych krawędzi, co generalnie nie jest oczywiste poniewaz rogów jest mniej.
- Duże rozbieżności pomiędzy wynikami metod PCA i t-SNE wynikają z błędnego ich użycia lub odczytania wyników przeze mnie

Wnioski na przyszłość:

- Pomimo ogromnej złożoności zależności tych danych można pokusić się o zbadanie wpływu kluczowych elementów na określony spadek czasu (x pomieszanych więcej krawędzi to y sekund ułożenia więcej)
- Sporządzenie podobnej analizy dla każdego bufora (bufor, element od którego rozpoczyna się zapamiętywanie kostki), aczkolwiek sądzę, że wyniki będą podobne, o ile nie identyczne.

Autor:
Krzysztof Bober, 3 semestr Analiza Danych, 386124

Wykorzystane źródła:

Principal Component Analysis (PCA) Algorithm - Amazon SageMaker
R Documentation and manuals | R Documentation
Materiały z zajęć dr Michała Seweryna

Link do kodu na Githubie:
Factor-analysis-for-3BLD-scrambles/solves_analysis.R at main · krzysztofbober/Factor-analysis-for-3BLD-scrambles (github.com)