Optimal Matching in Julia Using Distributed Computing

December 7, 2021

1 Optimal Matching in Julia Using Distributed Computing

Kamila Kolpashikova 2021

In time use research, one of the challenges is that we have considerably long sequences (96, 144, or 1440 steps), and we usually use optimal matching to calculate dissimilarity scores on those sequences. However, the optimal matching technique is very computationally demanding. At the moment, most social scientists use the TraMineR R package, which usually gives a great performance.

In this tutorial, I use Julia on an iCore9 Mac laptop (16 threads), which reduces the calculation time of 18776 sequences in half (compared to TraMineR).

You can follow the tutorial using the [code on GitHub](#)

```
[1]: ## change directory to the directory where you want to store the project
      cd("/Users/kamilakolpashnikova/Documents")
      home_path = pwd()
      home_path
```

```
[1]: "/Users/kamilakolpashnikova/Documents"
```

```
[2]: ## Load packages
      using Bio
      using Bio.Align
      using DataFrames
      using CSV
      using ZipFile
      using BenchmarkTools
      using StatsBase
      using Dates
```

```
[3]: using Distributed
      nprocs()
```

```
[3]: 1
```

```
[4]: import Hwloc
      n = Hwloc.num_physical_cores()
```

[4]: 8

I use a Mac with **iCore 9**. It has 16 threads, so I'm multiplying `n` (number of physical cores) by 2 to add the number of available threads. You need to check your own laptop's processor to figure out how many threads are available. If not sure, comment out the first line and uncomment the second, and you will use just the number of physical cores.

If you are using Mac, start Activity Monitor and open the CPU Load graph. It will show you the number of threads (Mac names them all 'cores').

```
[5]: addprocs(n*2, exeflags=`--project=$@__DIR__`)  
      #addprocs(n, exeflags=`--project=$@__DIR__`)  
      nprocs()
```

[5]: 17

```
[6]: @everywhere using Distributed  
      using DistributedArrays  
      @everywhere using DistributedArrays
```

```
[7]: #if any of the packages above do not work then run  
      #import Pkg; Pkg.add("DistributedArrays")
```

1.1 Download and Unzip the ATUS files

```
[8]: function open_file_to_dataframe(link, home_path)  
      download(link, home_path * last(split(link, "/"))  
      zarchive = ZipFile.Reader(home_path * last(split(link, "/"))  
  
      daf = DataFrame()  
  
      for file in zarchive.files  
          if last(split(file.name, ".")) == "dat"  
              #writing .dat file into home dir  
              write(home_path * "/" * file.name, read(file, String))  
  
              #reading the file and transforming into array of arrays  
              f = open(home_path * "/" * file.name);  
              lines = readlines(f)  
              lines = split.(lines, ',')  
  
              #transforming arr of arr into dataframe  
              namelist = Symbol.(lines[1])  
  
              for (i, name) in enumerate(namelist)  
                  daf[name] = [lines[j][i] for j in 2:length(lines)]  
              end  
          end  
      end
```

```

end
close(zarchive)
return daf
end

```

[8]: open_file_to_dataframe (generic function with 1 method)

```

[9]: link = "https://www.bls.gov/tus/special.requests/atusact-0320.zip"
      #download(link)
      df_full = open_file_to_dataframe(link, home_path)
      first(df_full, 5)

```

[9]:

	TUCASEID	TUACTIVITY_N	TUACTDUR24	TUCC5	TUCC5B	TRTCCTOT_LN	TRTC
	SubStri...	SubStri...	SubStri...	SubStri...	SubStri...	SubStri...	SubS
1	20030100013280	1	60	-1	-1	-1	-
2	20030100013280	2	30	-1	-1	-1	-
3	20030100013280	3	600	-1	-1	-1	-
4	20030100013280	4	150	-1	-1	-1	-
5	20030100013280	5	5	-1	-1	-1	-

```

[10]: df_ec = open_file_to_dataframe("https://www.bls.gov/tus/special.requests/
      ↪atusrosted-1120.zip",
      home_path)
      first(df_ec, 5)

```

[10]:

	TUCASEID	TEAGE_EC	TEELDUR	TEELWHO	TEELYRS	TRELHH	TUECLNO	TULIN
	SubStri...	SubStri...	SubStri...	SubStri...	SubStri...	SubStri...	SubStri...	SubS
1	20110101110074	70	4	44	2	0	5	-1
2	20110101110156	85	4	46	2	0	5	-1
3	20110101110507	80	1	55	-1	0	2	-1
4	20110101110521	85	3	43	-1	0	3	-1
5	20110101110522	80	4	44	6	0	2	-1

```

[11]: ## select only ids

      df_ec = select(df_ec, "TUCASEID")
      first(df_ec, 5)

```

[11]:

	TUCASEID
	SubStri...
1	20110101110074
2	20110101110156
3	20110101110507
4	20110101110521
5	20110101110522

```
[12]: ## filter caregivers

df = innerjoin(df_full, df_ec, on="TUCASEID")
df = unique(df)
nrow(df)
```

[12]: 395955

```
[13]: ## select only needed columns

df = select(df, ["TUCASEID", "TUACTIVITY_N", "TRCODEP", "TUACTDUR24"])
last(df, 5)
```

```
[13]:
```

	TUCASEID	TUACTIVITY_N	TRCODEP	TUACTDUR24
	SubStri...	SubStri...	SubStri...	SubStri...
1	20201212202312	16	020203	20
2	20201212202312	17	010201	30
3	20201212202312	18	120312	90
4	20201212202312	19	120303	120
5	20201212202312	20	010101	420

```
[14]: colnames = ["caseid", "actline", "activity", "duration"]
rename!(df, Symbol.(colnames))
first(df, 5)
```

```
[14]:
```

	caseid	actline	activity	duration
	SubStri...	SubStri...	SubStri...	SubStri...
1	20110101110074	1	110101	10
2	20110101110074	2	020402	300
3	20110101110074	3	180482	5
4	20110101110074	4	500101	525
5	20110101110074	5	180482	5

```
[15]: ## custom functions recoding activity codes (you need to change it
# if you want to focus on other activities)
# the coding is arbitrary (this is what I usually use in my research--
# you can change it the way you want)
#"1" or "a" = "Sleep",
#"2" or "b" = "Personal Care",
#"3" or "c" = "Housework",
#"4" or "d" = "Child Care",
#"5" or "e" = "Adult Care",
#"6" or "f" = "Work and Education",
#"7" or "g" = "Shopping",
#"8" or "h" = "TV Watching",
#"9" or "i" = "Eating",
#"10" or "j" = "Leisure",
#"11" = "Travel and Other"
```

```

function a_trans(val)
    a_dct = Dict{1 => "a", 2 => "b", 3 => "c", 4 => "d",
        5 => "e", 6 => "f", 7 => "g", 8 => "h", 9 => "i",
        10 => "j", 11 => "k"}
    return a_dct[val]
end

function activity_trans(val)
    act_dct = Dict{"010101" => 1, "010102" => 1,
        "010199" => 1, "010201" => 2, "010299" => 2,
        "010301" => 2, "010399" => 2, "010401" => 2,
        "010499" => 2, "010501" => 2, "010599" => 2,
        "019999" => 2, "020101" => 3, "020102" => 3,
        "020103" => 3, "020104" => 3, "020199" => 3,
        "020201" => 3, "020202" => 3, "020203" => 3,
        "020299" => 3, "020301" => 3, "020302" => 3,
        "020303" => 3, "020399" => 3, "020400" => 3,
        "020401" => 3, "020402" => 3, "020499" => 3,
        "020500" => 3, "020501" => 3, "020502" => 3,
        "020599" => 3, "020600" => 3, "020601" => 3,
        "020602" => 3, "020603" => 3, "020681" => 10,
        "020699" => 3, "020700" => 3, "020701" => 3,
        "020799" => 3, "020800" => 3, "020801" => 3,
        "020899" => 3, "020900" => 3, "020901" => 3,
        "020902" => 3, "020903" => 3, "020904" => 3,
        "020905" => 3, "020999" => 3, "029900" => 3,
        "029999" => 3, "030100" => 4, "030101" => 4,
        "030102" => 4, "030103" => 4, "030104" => 4,
        "030105" => 4, "030106" => 4, "030107" => 4,
        "030108" => 4, "030109" => 4, "030110" => 4,
        "030111" => 4, "030112" => 4, "030199" => 4,
        "030200" => 4, "030201" => 4, "030202" => 4,
        "030203" => 4, "030204" => 4, "030299" => 4,
        "030300" => 4, "030301" => 4, "030302" => 4,
        "030303" => 4, "030399" => 4, "040100" => 4,
        "040101" => 4, "040102" => 4, "040103" => 4,
        "040104" => 4, "040105" => 4, "040106" => 4,
        "040107" => 4, "040108" => 4, "040109" => 4,
        "040110" => 4, "040111" => 4, "040112" => 4,
        "040199" => 4, "040200" => 4, "040201" => 4,
        "040202" => 4, "040203" => 4, "040204" => 4,
        "040299" => 4, "040300" => 4, "040301" => 4,
        "040302" => 4, "040303" => 4, "040399" => 4,
        "030186" => 4, "040186" => 4, "030000" => 5,
        "030400" => 5, "030401" => 5, "030402" => 5,
    }

```

"030403" => 5, "030404" => 5, "030405" => 5,
 "030499" => 5, "030500" => 5, "030501" => 5,
 "030502" => 5, "030503" => 5, "030504" => 5,
 "030599" => 5, "039900" => 5, "039999" => 5,
 "040000" => 5, "040400" => 5, "040401" => 5,
 "040402" => 5, "040403" => 5, "040404" => 5,
 "040405" => 5, "040499" => 5, "040500" => 5,
 "040501" => 5, "040502" => 5, "040503" => 5,
 "040504" => 5, "040505" => 5, "040506" => 5,
 "040507" => 5, "040508" => 5, "040599" => 5,
 "049900" => 5, "049999" => 5, "050000" => 6,
 "050100" => 6, "050101" => 6, "050102" => 6,
 "050103" => 6, "050104" => 6, "050199" => 6,
 "050200" => 6, "050201" => 6, "050202" => 6,
 "050203" => 6, "050204" => 6, "050205" => 6,
 "050299" => 6, "050300" => 6, "050301" => 6,
 "050302" => 6, "050303" => 6, "050304" => 6,
 "050305" => 6, "050399" => 6, "050400" => 6,
 "050401" => 6, "050403" => 6, "050404" => 6,
 "050405" => 6, "050499" => 6, "059900" => 6,
 "059999" => 6, "060000" => 6, "060100" => 6,
 "060101" => 6, "060102" => 6, "060103" => 6,
 "060104" => 6, "060199" => 6, "060200" => 6,
 "060201" => 6, "060202" => 6, "060203" => 6,
 "060204" => 6, "060299" => 6, "060300" => 6,
 "060301" => 6, "060302" => 6, "060303" => 6,
 "060399" => 6, "060400" => 6, "060401" => 6,
 "060402" => 6, "060403" => 6, "060499" => 6,
 "069900" => 6, "069999" => 6, "050481" => 6,
 "050389" => 6, "050189" => 6, "060289" => 6,
 "050289" => 6, "070000" => 7, "070100" => 7,
 "070101" => 7, "070102" => 7, "070103" => 7,
 "070104" => 7, "070105" => 7, "070199" => 7,
 "070200" => 7, "070201" => 7, "070299" => 7,
 "070300" => 7, "070301" => 7, "070399" => 7,
 "079900" => 7, "079999" => 7, "080000" => 7,
 "080100" => 7, "080101" => 7, "080102" => 7,
 "080199" => 7, "080200" => 7, "080201" => 7,
 "080202" => 7, "080203" => 7, "080299" => 7,
 "080300" => 7, "080301" => 7, "080302" => 7,
 "080399" => 7, "080400" => 7, "080401" => 7,
 "080402" => 7, "080403" => 7, "080499" => 7,
 "080500" => 7, "080501" => 7, "080502" => 7,
 "080599" => 7, "080600" => 7, "080601" => 7,
 "080602" => 7, "080699" => 7, "080700" => 7,
 "080701" => 7, "080702" => 7, "080799" => 7,
 "080800" => 7, "080801" => 7, "080899" => 7,

"089900" => 7, "089999" => 7, "090000" => 7,
 "090100" => 7, "090101" => 7, "090102" => 7,
 "090103" => 7, "090104" => 7, "090199" => 7,
 "090200" => 7, "090201" => 7, "090202" => 7,
 "090299" => 7, "090300" => 7, "090301" => 7,
 "090302" => 7, "090399" => 7, "090400" => 7,
 "090401" => 7, "090402" => 7, "090499" => 7,
 "090500" => 7, "090501" => 7, "090502" => 7,
 "090599" => 7, "099900" => 7, "099999" => 7,
 "100000" => 7, "100100" => 7, "100101" => 7,
 "100102" => 7, "100103" => 7, "100199" => 7,
 "100200" => 7, "100201" => 7, "100299" => 7,
 "100300" => 7, "100303" => 7, "100304" => 7,
 "100399" => 7, "100400" => 7, "100401" => 7,
 "100499" => 7, "109900" => 7, "109999" => 7,
 "120303" => 8, "120304" => 8, "110000" => 9,
 "110100" => 9, "110101" => 9, "110199" => 9,
 "110200" => 9, "110201" => 9, "110299" => 9,
 "119900" => 9, "110289" => 9, "119999" => 9,
 "120000" => 10, "120100" => 10, "120101" => 10,
 "120199" => 10, "120200" => 10, "120201" => 10,
 "120202" => 10, "120299" => 10, "120300" => 10,
 "120301" => 10, "120302" => 10, "120305" => 10,
 "120306" => 10, "120307" => 10, "120308" => 10,
 "120309" => 10, "120310" => 10, "120311" => 10,
 "120312" => 10, "120313" => 10, "120399" => 10,
 "120400" => 10, "120401" => 10, "120402" => 10,
 "120403" => 10, "120404" => 10, "120405" => 10,
 "120499" => 10, "120500" => 10, "120501" => 10,
 "120502" => 10, "120503" => 10, "120504" => 10,
 "120599" => 10, "129900" => 10, "129999" => 10,
 "130000" => 10, "130100" => 10, "130101" => 10,
 "130102" => 10, "130103" => 10, "130104" => 10,
 "130105" => 10, "130106" => 10, "130107" => 10,
 "130108" => 10, "130109" => 10, "130110" => 10,
 "130111" => 10, "130112" => 10, "130113" => 10,
 "130114" => 10, "130115" => 10, "130116" => 10,
 "130117" => 10, "130118" => 10, "130119" => 10,
 "130120" => 10, "130121" => 10, "130122" => 10,
 "130123" => 10, "130124" => 10, "130125" => 10,
 "130126" => 10, "130127" => 10, "130128" => 10,
 "130129" => 10, "130130" => 10, "130131" => 10,
 "130132" => 10, "130133" => 10, "130134" => 10,
 "130135" => 10, "130136" => 10, "130199" => 10,
 "130200" => 10, "130201" => 10, "130202" => 10,
 "130203" => 10, "130204" => 10, "130205" => 10,
 "130206" => 10, "130207" => 10, "130208" => 10,

"130209" => 10, "130210" => 10, "130211" => 10,
 "130212" => 10, "130213" => 10, "130214" => 10,
 "130215" => 10, "130216" => 10, "130217" => 10,
 "130218" => 10, "130219" => 10, "130220" => 10,
 "130221" => 10, "130222" => 10, "130223" => 10,
 "130224" => 10, "130225" => 10, "130226" => 10,
 "130227" => 10, "130228" => 10, "130229" => 10,
 "130230" => 10, "130231" => 10, "130232" => 10,
 "130299" => 10, "130300" => 10, "130301" => 10,
 "130302" => 10, "130399" => 10, "130400" => 10,
 "130401" => 10, "130402" => 10, "130499" => 10,
 "139900" => 10, "139999" => 10, "140000" => 10,
 "140100" => 10, "140101" => 10, "140102" => 10,
 "140103" => 10, "140104" => 10, "140105" => 10,
 "149900" => 10, "149999" => 10, "150000" => 10,
 "150100" => 10, "150101" => 10, "150102" => 10,
 "150103" => 10, "150104" => 10, "150105" => 10,
 "150106" => 10, "150199" => 10, "150200" => 10,
 "150201" => 10, "150202" => 10, "150203" => 10,
 "150204" => 10, "150299" => 10, "150300" => 10,
 "150301" => 10, "150302" => 10, "150399" => 10,
 "150400" => 10, "150401" => 10, "150402" => 10,
 "150499" => 10, "150500" => 10, "150501" => 10,
 "150599" => 10, "150600" => 10, "150601" => 10,
 "150602" => 10, "150699" => 10, "150700" => 10,
 "150701" => 10, "150799" => 10, "150800" => 10,
 "150801" => 10, "150899" => 10, "159900" => 10,
 "159999" => 10, "160000" => 10, "160100" => 10,
 "160101" => 10, "160102" => 10, "160103" => 10,
 "160104" => 10, "160105" => 10, "160106" => 10,
 "160107" => 10, "160108" => 10, "160199" => 10,
 "160200" => 10, "160201" => 10, "160299" => 10,
 "169900" => 10, "169999" => 10, "159989" => 10,
 "169989" => 10, "110281" => 10, "100381" => 10,
 "100383" => 10, "180000" => 11, "180100" => 11,
 "180101" => 11, "180199" => 11, "180200" => 11,
 "180201" => 11, "180202" => 11, "180203" => 11,
 "180204" => 11, "180205" => 11, "180206" => 11,
 "180207" => 11, "180208" => 11, "180209" => 11,
 "180280" => 11, "180299" => 11, "180300" => 11,
 "180301" => 11, "180302" => 11, "180303" => 11,
 "180304" => 11, "180305" => 11, "180306" => 11,
 "180307" => 11, "180399" => 11, "180400" => 11,
 "180401" => 11, "180402" => 11, "180403" => 11,
 "180404" => 11, "180405" => 11, "180406" => 11,
 "180407" => 11, "180482" => 11, "180499" => 11,
 "180500" => 11, "180501" => 11, "180502" => 11,


```

"180503" => 11, "180504" => 11, "180599" => 11,
"180600" => 11, "180601" => 11, "180602" => 11,
"180603" => 11, "180604" => 11, "180605" => 11,
"180699" => 11, "180700" => 11, "180701" => 11,
"180702" => 11, "180703" => 11, "180704" => 11,
"180705" => 11, "180782" => 11, "180799" => 11,
"180800" => 11, "180801" => 11, "180802" => 11,
"180803" => 11, "180804" => 11, "180805" => 11,
"180806" => 11, "180807" => 11, "180899" => 11,
"180900" => 11, "180901" => 11, "180902" => 11,
"180903" => 11, "180904" => 11, "180905" => 11,
"180999" => 11, "181000" => 11, "181001" => 11,
"181002" => 11, "181099" => 11, "181100" => 11,
"181101" => 11, "181199" => 11, "181200" => 11,
"181201" => 11, "181202" => 11, "181203" => 11,
"181204" => 11, "181205" => 11, "181206" => 11,
"181283" => 11, "181299" => 11, "181300" => 11,
"181301" => 11, "181302" => 11, "181399" => 11,
"181400" => 11, "181401" => 11, "181499" => 11,
"181500" => 11, "181501" => 11, "181599" => 11,
"181600" => 11, "181601" => 11, "181699" => 11,
"181800" => 11, "181801" => 11, "181899" => 11,
"189900" => 11, "189999" => 11, "180481" => 11,
"180381" => 11, "180382" => 11, "181081" => 11,
"180589" => 11, "180682" => 11, "500000" => 11,
"500100" => 11, "500101" => 11, "500102" => 11,
"500103" => 11, "500104" => 11, "500105" => 11,
"500106" => 11, "500107" => 11, "509900" => 11,
"509989" => 11, "509999" => 11)

return a_trans(act_dct[val])
end

```

[15]: activity_trans (generic function with 1 method)

I avoid using the numbers for sequence coding because the way how the pairalign function in Bio.jl package works. It takes strings as sequences, and if the numbers are used and the number of activities is over 9, there is a problem of separating activities. There are ways around it, but I chose to use characters instead.

```

[16]: df["act"] = activity_trans.(df["activity"])
first(df,5)

```

[16]:

	caseid	actline	activity	duration	act
	SubStri...	SubStri...	SubStri...	SubStri...	String
1	20110101110074	1	110101	10	i
2	20110101110074	2	020402	300	c
3	20110101110074	3	180482	5	k
4	20110101110074	4	500101	525	k
5	20110101110074	5	180482	5	k

```
[17]: ## check if all activities were recoded. If numbers appear in the list
# --> some of the activities were missed
```

```
unique(df["act"])
```

```
[17]: 11-element Array{String,1}:
```

```
"i"
"c"
"k"
"b"
"h"
"a"
"j"
"e"
"g"
"d"
"f"
```

```
[18]: ## change the type of activity duration variable to integers
```

```
df[:dur] = parse.(Int64, df["duration"])
first(df, 5)
```

```
[18]:
```

	caseid	actline	activity	duration	act	dur
	SubStri...	SubStri...	SubStri...	SubStri...	String	Int64
1	20110101110074	1	110101	10	i	10
2	20110101110074	2	020402	300	c	300
3	20110101110074	3	180482	5	k	5
4	20110101110074	4	500101	525	k	525
5	20110101110074	5	180482	5	k	5

```
[19]: ## create sequences separately for each activity line
```

```
df["act_rep"] = df["act"]
for i in 1:nrow(df)
    df["act_rep"][i] = repeat(df["act"][i], df["dur"][i])
end
first(df, 3)
```

```
[19]:
```

	caseid	actline	activity	duration	
	SubStri...	SubStri...	SubStri...	SubStri...	
1	20110101110074	1	110101	10	
2	20110101110074	2	020402	300	cc
3	20110101110074	3	180482	5	

```
[20]: ## to combine the sequences by caseid,
      # I'll create a separate dictionary with caseid's as keys
```

```
tempoTable = Dict()
for i in unique(df["caseid"])
    tempoTable[i] = []
end
```

```
[21]: ## function that will divide an array (sequence) into equal parts of length n
```

```
chunk(arr, n) = [arr[i:min(i + n - 1, end)] for i in 1:n:length(arr)]
```

```
[21]: chunk (generic function with 1 method)
```

```
[22]: ## in these lines I loop over the df grouped by caseid
      # collect all act_rep per caseid into sequences per caseid
      # divide sequences into chunks of 15 (representing 15 minutes)
      # choose the most common activity code to represent the chunk
```

```
for d in groupby(df, :caseid)
    temp = join(push!(tempoTable[d["caseid"]][1], d["act_rep"])[1])
    lst = ""
    for c in chunk(temp, 15)
        lst = lst * first(countmap(c))[1]
    end
    tempoTable[d["caseid"]][1] = lst
end
```

```
[23]: tempoTable
```

```
[23]: Dict{Any,Any} with 18776 entries:
```

```
"20120302120003" => "aaaaaaaaaaciuhhhhhhhhhkkggggkccciuhhhhhkkkggggkkcciii...
"20180806182262" => "aaaaaaaaaaaaaaaaaijkkkkkggggggggggggkkkkkiuhhhhhhhhjjjj...
"20141211142443" => "aaaaaaaaaaaaaaaaajdddddkkkeekkkkkkkkjjjjjjjjjjjjjjiikk...
"20141109142047" => "aaaaaaaaaaaaaaaaabbbbbbdkddjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj...
"20140504141362" => "aaaaaaaaaacbbbkfffffffffffffffffffffffffffffffffffffffff...
"20140112131728" => "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaajcccccccccccccccccccccccc...
"20150807150158" => "aaaaaaibbfffffjffffffffffjffffffffffjffffffffffjffffffffffkhh...
"20201110201074" => "aaaaaaaaadddcccckggggggggkggcccccccccbdddddjjjjkdk...
"20150112140644" => "aaaaaaabiifffffffffffffffffffffffffffffkkkkkkkkkggggkkccccc...
"20150403151136" => "hhccciijjaaaaaajjjhhhhhhhhhhchhccciijhhhhhhhhhhccj...
```

```

"20110302110628" => "aaaaaaaaaaaaaaaaabjjjjjjhhhhcihhhhhhhhhhhaaaaaaaaaajj...
"20170111160007" => "aaaaaaaaaaaaaaaaaddcdckjjjjjjjjjjjjjjjjjjjjjjjjjjkkcc...
"20200605201979" => "aaaaaaaaaaaaaaaaaciicccjjccccccciicjjjjjjjjbbbbbjjjjj...
"20180403181820" => "aaaaaaaaaaaaaaaaaciiiiicccccccckgggggggkccccchhhhi...
"20191110191403" => "aaaaaaaajjjcciiicccbbbbbkkjjjjjjjjkkcccccccccccccccccc...
"20120302120660" => "aaaaaaaaaaaaabbijjjccjjjjjjjjhhhhhhhhhhhhhhcccccihhhh...
"20130302131716" => "aaaaaaaajjjbbkkkkkgkjjjjjjjjjjjjjjjjjjkkccijjjbkjjjjjjj...
"20131110131665" => "aaaaaaaaaaaaabbbccccccciijjjjjjjjjjjjjjjjjjjjjcjjjjjj...
"20191009190108" => "aaaaaaaaikkkkbjjjjjjjjjjjjjjjjjjjjjjjjjjjccccccccccjcjj...
"20171010172082" => "aaaaaaaaaaaaaaaaacibhbkjjjjjjjjjjjjjjjjjjjjgggkkggggccc...
"20160302161799" => "aaaaaaaaaaaaajjjjjjjjjjjjjjjbbbggggggggggggggiiddkkggggk...
"20160806162407" => "aaaaaaaaaaaaaaaaaaaaaaaaahhhhhhhhjccccccckigggggggggcccjj...
"20181211180506" => "aaaaaaaadjckkkkjjjjjgggkcfccccccccccccccciicckkccccjjjj...
"20110403112514" => "aaaaaaaaaaaaaaaaaaaaaaaaakkkkkkkkkkggggggggggggkjjjjjjjjkh...
"20140605140679" => "aaaaaaaaaaaaaaaaaaaaaaaaeeeeeeffffffcciffffffffccccccch...
=>

```

```

[24]: ## when we transform sequences of 1440 minutes into 15-minutes slots
# the resulting sequences should be 96 steps in length
# these lines check if all values in tempoTable sequences are of length 96
# if it doesn't print anything it means that we are OK

```

```

for i in keys(tempoTable)
    if length(tempoTable[i]) != 96
        print(tempoTable[i])
        print(" ")
    end
end

```

```

[25]: ## let's create an array that contains only the sequences

```

```

lines = collect(values(tempoTable))

```

```

[25]: 18776-element Array{Any,1}:

```

```

"aaaaaaaaaciihhhhhhhhhkkggggkcciihhhhkkkggggkcciiihhhjjjjhhbbbbbbeehaaaaa
aaaaaaaaaaaaaaaaaaaa"
"aaaaaaaaaaaaaaaaaijjkkkkkggggggggggggkkkkiihhhhhhhhjjjjjjjjccijjjjjjjjjjjihj
aaaaaaaaaaaaaaaaaaaa"
"aaaaaaaaaaaaaaaaajdddddkkkekkkkkkkkjjjjjjjjjjjjjjiiikkkkkkkkkkkdddddhaaaaaaaa
aaaaaaaaaaaaaaaaaaaa"
"aaaaaaaaaaaaaaaaabbbbbbkkddjjjjjjjjiiigggdddddskkdkiiiiijjjjjjjjjjjjjjj
jjkaaaaaaaaaaaaaaa"
"aaaaaaaaaacbbkkfccccccccccccccccccccccccccccccccccccccccccciiiiibaaaaa
aaaaaaaaaaaaaaaaaaaa"
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaajcccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccjjjj"
"aaaaaibbfffffjffffffffffjffffffffffjffffffffffkhhhhhhhhhhhhhhhhhhhhhaa

```

[illegible]

```
[26]: ## it might be that you will be running OM again, so better save the lines
```

→ *variable*

```
outfile = "lines.txt"
open(outfile, "w") do f
  for i in lines
    println(f, i)
  end
end
```

```
[27]: ## you can then restart here, by just opening the lines file
@everywhere f = open("lines.txt");
@everywhere lines = readlines(f)
```

```
[28]: ## before we can distribute the tasks to our threads
# we need to give them all the data and packages
# that they need to perform the tasks
@everywhere using Bio.Align
@everywhere costmodel = CostModel(match=0, mismatch=2, insertion=1, deletion=1);
```

```
[29]: ## let's create a dissimilarity matrix where the results will be stored
## because I will be hashing the line codes
# I use UInt32 for preserving memory
# However, do not run more than 20000 sequences at once
# --> may run into memory problems
# and hashing problems (above 40000)

d_m = zeros(UInt32, length(lines), length(lines))
```

```
[29]: 18776×18776 Array{UInt32,2}:
 0x00000000 0x00000000 0x00000000 ... 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 ... 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 ... 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 ... 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 ... 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
 0x00000000 0x00000000 0x00000000 ... 0x00000000 0x00000000 0x00000000
```

```
[30]: ## hash the combinations of sequences that will be matched
      # using a simple integer representing first sequence + second sequence

      for i in 1:length(lines)
          for j in 1:length(lines)
              d_m[i,j] = i*100000 + j
          end
      end
end
```

```
[31]: ## transform dissimilarity matrix to a distributed array
```

```
A = DArray{I->d_m, (length(lines), length(lines))}
```

```
[31]: 18776×18776 DArray{UInt32,2,Array{UInt32,2}}:
0x000186a1 0x000186a2 0x000186a3 ... 0x000198f4 0x000198f5 0x000198f6
0x00030d41 0x00030d42 0x00030d43 ... 0x00031f94 0x00031f95 0x00031f96
0x000493e1 0x000493e2 0x000493e3 ... 0x0004a634 0x0004a635 0x0004a636
0x00061a81 0x00061a82 0x00061a83 ... 0x00062cd4 0x00062cd5 0x00062cd6
0x0007a121 0x0007a122 0x0007a123 ... 0x0007b374 0x0007b375 0x0007b376
0x000927c1 0x000927c2 0x000927c3 ... 0x00093a14 0x00093a15 0x00093a16
0x000aae61 0x000aae62 0x000aae63 ... 0x000ac0b4 0x000ac0b5 0x000ac0b6
0x000c3501 0x000c3502 0x000c3503 ... 0x000c4754 0x000c4755 0x000c4756
0x000dbba1 0x000dbba2 0x000dbba3 ... 0x000dcd4 0x000dcd5 0x000dcd6
0x000f4241 0x000f4242 0x000f4243 ... 0x000f5494 0x000f5495 0x000f5496
0x0010c8e1 0x0010c8e2 0x0010c8e3 ... 0x0010db34 0x0010db35 0x0010db36
0x00124f81 0x00124f82 0x00124f83 ... 0x001261d4 0x001261d5 0x001261d6
0x0013d621 0x0013d622 0x0013d623 ... 0x0013e874 0x0013e875 0x0013e876

0x1be9b0e1 0x1be9b0e2 0x1be9b0e3 ... 0x1be9c334 0x1be9c335 0x1be9c336
0x1beb3781 0x1beb3782 0x1beb3783 ... 0x1beb49d4 0x1beb49d5 0x1beb49d6
0x1becbe21 0x1becbe22 0x1becbe23 ... 0x1becd074 0x1becd075 0x1becd076
0x1bee44c1 0x1bee44c2 0x1bee44c3 ... 0x1bee5714 0x1bee5715 0x1bee5716
0x1befcb61 0x1befcb62 0x1befcb63 ... 0x1befddb4 0x1befddb5 0x1befddb6
0x1bf15201 0x1bf15202 0x1bf15203 ... 0x1bf16454 0x1bf16455 0x1bf16456
0x1bf2d8a1 0x1bf2d8a2 0x1bf2d8a3 ... 0x1bf2eaf4 0x1bf2eaf5 0x1bf2eaf6
0x1bf45f41 0x1bf45f42 0x1bf45f43 ... 0x1bf47194 0x1bf47195 0x1bf47196
0x1bf5e5e1 0x1bf5e5e2 0x1bf5e5e3 ... 0x1bf5f834 0x1bf5f835 0x1bf5f836
0x1bf76c81 0x1bf76c82 0x1bf76c83 ... 0x1bf77ed4 0x1bf77ed5 0x1bf77ed6
0x1bf8f321 0x1bf8f322 0x1bf8f323 ... 0x1bf90574 0x1bf90575 0x1bf90576
0x1bfa79c1 0x1bfa79c2 0x1bfa79c3 ... 0x1bfa8c14 0x1bfa8c15 0x1bfa8c16
```

```
[32]: ## function that will run optimal matching
      # 1. it will read the value and transform it into indexes of lines array
      # 2. it will run optimal matching on those two lines (sequences)
      # 3. only lower triangle of the dissimilarity matrix will be calculated
      # (for optimization purposes)
```

```

@everywhere function pairalignEditNum_Opt(num::UInt32,
    lines = lines, editdist = EditDistance(),
    cost = costmodel)
    pos1 = convert(UInt32, floor(num/100000))
    pos2 = num - pos1*100000
    if pos1>pos2
        return distance(pairalign(editdist, lines[pos1], lines[pos2], cost))
    else
        return 0
    end
end
end

```

```
[33]: ## log the start of OM
```

```
Dates.format(now(), "HH:MM")
```

```
[33]: "14:06"
```

```
[34]: output_mat_opt = pairalignEditNum_Opt.(A)
```

```
[34]: 18776×18776 DArray{Int64,2,Array{Int64,2}}:
```

```

 0  0  0  0  0  0  0  0 ...  0  0  0  0  0  0  0
84  0  0  0  0  0  0  0  0  0  0  0  0  0  0
100 70  0  0  0  0  0  0  0  0  0  0  0  0  0
108 76 80  0  0  0  0  0  0  0  0  0  0  0  0
100 120 114 114  0  0  0  0  0  0  0  0  0  0  0
136 136 136 136 136  0  0  0 ...  0  0  0  0  0  0  0
 94 116 126 134  56 140  0  0  0  0  0  0  0  0  0
 94 102  84  98 114 136 130  0  0  0  0  0  0  0  0
100 110 112 126  60 138  64 106  0  0  0  0  0  0  0
 78 102 110 128 124 136  90 110  0  0  0  0  0  0  0
 78  80  84  92 122 126  84 116 ...  0  0  0  0  0  0  0
 94  76  70  74 110 136 116 102  0  0  0  0  0  0  0
 94  72  80  84 110 110 114  96  0  0  0  0  0  0  0

 80 102 106 130 110 136  82 122  0  0  0  0  0  0  0
 82  82  80  94 114 136 120  92 ...  0  0  0  0  0  0  0
106 124 108 130  40 136  50 112  0  0  0  0  0  0  0
 92  94 106 114 132 136 104 114  0  0  0  0  0  0  0
 80  84  80  92 118 134  82 116  0  0  0  0  0  0  0
100  78  78  82 112 112 130  86  0  0  0  0  0  0  0
104  90  88  88 116 146 120 124 ... 90  0  0  0  0  0  0
 72  74  78  88 122 136 118  84  76 100  0  0  0  0  0
 88  66  60  68 112 136 124  96  86  80  72  0  0  0  0
 98 108 130 124  56 140  28 132 124 118 114 122  0  0  0
114 128 108 110 120 124 132  66 104 138 104 104 134  0  0
 74  94 106 114 110 124  98 106 ... 94 118  80  86 102 112  0

```



```
[35]: ## log the end of OM
```

```
Dates.format(now(), "HH:MM")
```

```
[35]: "14:32"
```

```
[36]: ## let's bring the distributed array back
```

```
B = Array{UInt32}(output_mat_opt)
```

```
[36]: 18776×18776 Array{UInt32,2}:
```

```
0x00000000 0x00000000 0x00000000 ... 0x00000000 0x00000000 0x00000000
0x00000054 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000064 0x00000046 0x00000000 0x00000000 0x00000000 0x00000000
0x0000006c 0x0000004c 0x00000050 0x00000000 0x00000000 0x00000000
0x00000064 0x00000078 0x00000072 0x00000000 0x00000000 0x00000000
0x00000088 0x00000088 0x00000088 ... 0x00000000 0x00000000 0x00000000
0x0000005e 0x00000074 0x0000007e 0x00000000 0x00000000 0x00000000
0x0000005e 0x00000066 0x00000054 0x00000000 0x00000000 0x00000000
0x00000064 0x0000006e 0x00000070 0x00000000 0x00000000 0x00000000
0x0000004e 0x00000066 0x0000006e 0x00000000 0x00000000 0x00000000
0x0000004e 0x00000050 0x00000054 ... 0x00000000 0x00000000 0x00000000
0x0000005e 0x0000004c 0x00000046 0x00000000 0x00000000 0x00000000
0x0000005e 0x00000048 0x00000050 0x00000000 0x00000000 0x00000000

0x00000050 0x00000066 0x0000006a 0x00000000 0x00000000 0x00000000
0x00000052 0x00000052 0x00000050 ... 0x00000000 0x00000000 0x00000000
0x0000006a 0x0000007c 0x0000006c 0x00000000 0x00000000 0x00000000
0x0000005c 0x0000005e 0x0000006a 0x00000000 0x00000000 0x00000000
0x00000050 0x00000054 0x00000050 0x00000000 0x00000000 0x00000000
0x00000064 0x0000004e 0x0000004e 0x00000000 0x00000000 0x00000000
0x00000068 0x0000005a 0x00000058 ... 0x00000000 0x00000000 0x00000000
0x00000048 0x0000004a 0x0000004e 0x00000000 0x00000000 0x00000000
0x00000058 0x00000042 0x0000003c 0x00000000 0x00000000 0x00000000
0x00000062 0x0000006c 0x00000082 0x00000000 0x00000000 0x00000000
0x00000072 0x00000080 0x0000006c 0x00000086 0x00000000 0x00000000
0x0000004a 0x0000005e 0x0000006a ... 0x00000066 0x00000070 0x00000000
```

```
[37]: for i in 1:convert(UInt32, length(B)^0.5)
      for j in 1:convert(UInt32, length(B)^0.5)
        if i<j
          if B[i,j] == 0
            B[i,j] = B[j,i]
          end
        end
      end
    end
  end
end
```

```
[38]: ## here is the final dissimilarity matrix
```

```
B
```

```
[38]: 18776×18776 Array{UInt32,2}:
 0x00000000 0x00000054 0x00000064 ... 0x00000062 0x00000072 0x0000004a
 0x00000054 0x00000000 0x00000046 0x0000006c 0x00000080 0x0000005e
 0x00000064 0x00000046 0x00000000 0x00000082 0x0000006c 0x0000006a
 0x0000006c 0x0000004c 0x00000050 0x0000007c 0x0000006e 0x00000072
 0x00000064 0x00000078 0x00000072 0x00000038 0x00000078 0x0000006e
 0x00000088 0x00000088 0x00000088 ... 0x0000008c 0x0000007c 0x0000007c
 0x0000005e 0x00000074 0x0000007e 0x0000001c 0x00000084 0x00000062
 0x0000005e 0x00000066 0x00000054 0x00000084 0x00000042 0x0000006a
 0x00000064 0x0000006e 0x00000070 0x0000003e 0x00000072 0x00000076
 0x0000004e 0x00000066 0x0000006e 0x00000064 0x00000070 0x0000005a
 0x0000004e 0x00000050 0x00000054 ... 0x0000005e 0x0000007c 0x00000046
 0x0000005e 0x0000004c 0x00000046 0x00000076 0x0000006e 0x00000058
 0x0000005e 0x00000048 0x00000050 0x00000072 0x0000005a 0x00000050

 0x00000050 0x00000066 0x0000006a 0x00000058 0x00000080 0x0000004a
 0x00000052 0x00000052 0x00000050 ... 0x00000074 0x00000074 0x00000058
 0x0000006a 0x0000007c 0x0000006c 0x00000034 0x00000076 0x0000006a
 0x0000005c 0x0000005e 0x0000006a 0x00000070 0x00000074 0x00000064
 0x00000050 0x00000054 0x00000050 0x0000005c 0x0000007a 0x00000054
 0x00000064 0x0000004e 0x0000004e 0x0000007c 0x00000068 0x0000005e
 0x00000068 0x0000005a 0x00000058 ... 0x00000076 0x0000008a 0x00000076
 0x00000048 0x0000004a 0x0000004e 0x00000072 0x00000068 0x00000050
 0x00000058 0x00000042 0x0000003c 0x0000007a 0x00000068 0x00000056
 0x00000062 0x0000006c 0x00000082 0x00000000 0x00000086 0x00000066
 0x00000072 0x00000080 0x0000006c 0x00000086 0x00000000 0x00000070
 0x0000004a 0x0000005e 0x0000006a ... 0x00000066 0x00000070 0x00000000
```

```
[39]: CSV.write("dissimilarity_matrix.csv", Tables.table(B), writeheader=false)
      ## you might also want to bind the caseid with this file -- I'll let you
      # figure it out by yourself
```

```
[39]: "dissimilarity_matrix.csv"
```

```
[40]: ## the file will be saved in the working directory storing the dissimilarity_
      ↪ scores matrix
      ## you can use that matrix for further analysis either in Julia or other_
      ↪ languages (Python, R)
```

Good luck with your research!

1.1.1 References

1. Kolpashnikova, K. (2021, November 10). Sequence Analysis: Time Use Data (ATUS) in R.
<https://doi.org/10.31219/osf.io/ts34v>