Analysis of one interesting alliance graph within tidy network analysis framework.

**Prologue**

Quite some time ago I read the fantastic [“Tao Te Programming”](http://www.burns-stat.com/documents/books/tao-te-programming/) book by [Patrick Burns](http://www.burns-stat.com/). You can know this author from his everlasting work [“The R Inferno”](http://www.burns-stat.com/documents/books/the-r-inferno/).

“Tao Te Programming” is a vision about what a good programming should be. It is written in [programming] language-agnostic fashion but mainly with R in mind. It is organized in 81 chapters with some distinctive feature: at the end of the most chapters there are lists of chapters-“allies” and chapters-“opponents” to the chapter in question. During the reading I was really interested in the properties of [graph](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)) that is created with these connections.

From the other side of my life, I constantly hear about packages for tidy network analysis: tidygraph & ggraph. Both created by “compulsive package developer” (not sure, where to put “great” in this description) [Thomas Lin Pedersen](https://www.data-imaginist.com/). They provide tidy tools for creating, manipulating and plotting network data. It happened, that I didn’t have an opportunity to work with them… until now.

**Overview**

This post is mainly about exploration of “Tao Te Programming” alliance graph using tidygraph and ggraph, with a few descriptions about those package functionality (for a thorough overview check out this [speach](https://www.rstudio.com/resources/videos/tidying-up-your-network-analysis-with-tidygraph-and-ggraph/) by Thomas at rstudio::conf 2018). It will cover:

* **Creation**, **visualization** and **exploration** of alliance graph.
* Determination of “central programming advice” or **“Why you should learn to dance the Debug 2-Step”**.

Setup:

library(taoteprog)

library(tidyverse)

library(tidygraph)

library(ggraph)

set.seed(201803)

**Alliance graph**

**Creation**

Alliance graph is a set of chapters-nodes which created based on “ally” and “opponent” connections. It is created using the following data from taoteprog (which I manually created while reading a book):

* ttp\_chapters – a [tibble](http://tibble.tidyverse.org/) with chapter data (column chapter for its number and name for its name). The whole list of chapters can be found [here](http://www.burns-stat.com/documents/books/tao-te-programming/table-of-contents/).
* ttp\_edges – a tibble with data about chapter connections. It has the following columns:
  + chapter1 for the chapter number at which list of connections is printed.
  + chapter2 for the number of connected chapter.
  + type for connection type (either “ally” or “opponent”).

**Note** that connections between chapters are not symmetrical, i.e. one chapter can be an “ally” or “opponent” of the another but not the other way around.

ttp\_raw <- tbl\_graph(nodes = ttp\_chapters, edges = ttp\_edges, directed = TRUE)

ttp\_raw

## # A tbl\_graph: 81 nodes and 124 edges

## #

## # A directed multigraph with 20 components

## #

## # Node Data: 81 x 2 (active)

## chapter name

##

## 1 1 Program

## 2 2 Program Well

## 3 3 Think Chess

## 4 4 Carve Reality

## 5 5 Solve the Problem

## 6 6 Don't Solve the Problem

## # ... with 75 more rows

## #

## # Edge Data: 124 x 3

## from to type

##

## 1 4 9 ally

## 2 4 54 ally

## 3 5 6 opponent

## # ... with 121 more rows

The main data structure of tidygraph is a tbl\_graph which can be thought of as a combination of tibbles with data for nodes and edges. **Note** that values in from and to columns in edges tibble are the row indices of the nodes stored in nodes tibble, which will be a little confusing when nodes won’t cover all chapters.

Important features of this graph:

* There are 2 loops in graph (chapters “Be Consistent”, “Follow The Way”). And all of them are of type “opponent”, which I consider to be a subtle joke rather than editor mistake. However, for exploration these edges will be removed.
* There are 20 [weak] components in this graph (maximal groups of nodes where each pair can be connected by undirected path). The number is this big because there are many chapters with no “allies” and no “opponents” (isolated points in graph), which will be also removed for exploration.

So the graph of interest is constructed as follows:

ttp <- ttp\_raw %>%

# Remove loops

activate(edges) %>%

filter(!edge\_is\_loop()) %>%

# Remove isolated nodes

activate(nodes) %>%

filter(!node\_is\_isolated())

ttp

## # A tbl\_graph: 67 nodes and 122 edges

## #

## # A directed simple graph with 6 components

## #

## # Node Data: 67 x 2 (active)

## chapter name

##

## 1 4 Carve Reality

## 2 5 Solve the Problem

## 3 6 Don't Solve the Problem

## 4 7 Enjoy Confusion

## 5 8 Procrastinate

## 6 9 Verbalize and Nounalize

## # ... with 61 more rows

## #

## # Edge Data: 122 x 3

## from to type

##

## 1 1 6 ally

## 2 1 44 ally

## 3 2 3 opponent

## # ... with 119 more rows

Some explanations of the magic just happened:

* activate() verb from tidygraph serves as a switch between tibbles for nodes and edges. All [dplyr](http://dplyr.tidyverse.org/) verbs applied to tbl\_graph object are applied to the active tibble.
* edge\_is\_loop(), node\_is\_isolated() and group\_components() are functions of the same essence as n() in dplyr: they both should be called inside a graph computation functions.

**Visualization**

ttp %>%

ggraph(layout = "nicely") +

geom\_edge\_link(

aes(colour = type),

arrow = arrow(length = unit(1.5, "mm")),

start\_cap = circle(3, "mm"),

end\_cap = circle(3, "mm")

) +

geom\_node\_text(aes(label = chapter), size = 5) +

scale\_edge\_colour\_manual(values = c(ally = "#22B022",

opponent = "#A4AAF6")) +

theme\_graph() +

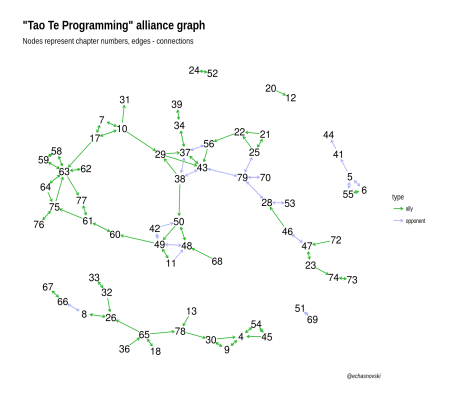
labs(

title = '"Tao Te Programming" alliance graph',

subtitle = "Nodes represent chapter numbers, edges - connections",

caption = "@echasnovski"

)



Important notes about creation of this plot:

* ggraph() is the equivalent of ggplot() in [ggplot2](http://ggplot2.tidyverse.org/) and also returns a ggplot object. ggraph() takes a layout argument for the type of graph arrangement in plot.
* geom\_edge\_link() and geom\_node\_text() are very much ggplot2-like functions: they create layers of plot (for links and nodes respectively).
* theme\_graph() is a pre-installed ggraph theme.

**Exploration**

Notes about graph structure:

There are two relatively big components (with 16 and 40 nodes) and 4 small ones (one with 5 nodes and three with 2).

There are considerably more “ally” connections then “opponent” which can be confirmed by the following code:

ttp\_edges %>% count(type)

## # A tibble: 2 x 2

## type n

##

## 1 ally 88

## 2 opponent 36

The graph itself can be considered as [signed graph](https://en.wikipedia.org/wiki/Signed_graph), i.e. a graph “in which each edge has a positive or negative sign”. This type of graphs can be studied for special community detection: nodes within common community should be primarily connected with positive (“ally”) edges and between communities – by negative (“opponent”) ones.

Interesting problem description and solution are presented in this [paper](http://mrvar.fdv.uni-lj.si/pajek/SignedNetworks/Bled94.pdf). Implementation is rather verbose with no very special insight, so I decided to not include it here.

**Central advice**

One of the reasons I started this research is to find “central programming advice” based on the “Tao Te Programming”. By that I mean a piece of advice as the core wisdom which should be mastered first as other advice “depend” on it.

Perhaps, one of the most popular way to compute the centrality value of graph nodes is [PageRank](https://en.wikipedia.org/wiki/PageRank) algorithm. The core idea is that edges of the graph are treated as possibilities for transition to another node, which current node “sees” as important. PageRank assumes that some traveler starts at random node and constantly moves using “available edges”. Sometimes, the process is restarted at the new (randomly picked) starting node and transition continues. The **PageRank score of the node** is the probability of finding a traveler in a particular node at random time in the future: the more this value the more important this node.

This approach needs some correction to be applied to our alliance graph because present edges are not all the same. “Ally” edges can be considered as normal edges in the described framework but what should we do with “opponent” edges? We will go by the following logic: if “ally” edge makes more important node **to which** it is headed, then “opponent” edge should make more important node **from which** it is headed. This can be accomplished by introducing “peaceful” variation of alliance graph: all “opponent” edges are reversed and tagged as “ally”. This way starting node of [previously] “opponent” edge will receive more possibilities for traveler to appear in it.

Using “peaceful” alliance graph, **constructed from all chapters and edges** (for more proximity to raw data), we can determine nodes with the highest PageRank scores with the following code:

edges\_peace <- ttp\_edges %>%

transmute(

from = if\_else(type == "ally", chapter1, chapter2),

to = if\_else(type == "ally", chapter2, chapter1)

)

tbl\_graph(

nodes = ttp\_chapters, edges = edges\_peace, directed = TRUE

) %>%

activate(nodes) %>%

mutate(centr = centrality\_pagerank()) %>%

as\_tibble() %>%

arrange(desc(centr))

## # A tibble: 81 x 3

## chapter name centr

##

## 1 63 Dance the Debug 2-Step 0.0403

## 2 4 Carve Reality 0.0346

## 3 61 Think Safety 0.0324

## 4 8 Procrastinate 0.0313

## 5 37 Become a Ghost 0.0291

## # ... with 76 more rows

The most important advice is from chapter 63 “Dance the Debug 2-Step”. This is a rather long chapter which contains many peaces of mini-advice about debugging process. My personal favorite:

* **The Debug 2-step** is 1) DON’T PANIC!! and 2) Figure out what’s wrong. Not very easy steps.
* The fastest way to debug is to **not have bugs that are hard to find**.
* One of the best ways to track down a bug is to **continuously simplify the problem**.
* Simplification should be **driven by hypotheses you make**. The cycle is: 1) create a hypothesis, 2) test the hypothesis.
* A bug means there is at least one assumption that is being violated. You are really **looking for false assumptions**.
* **The harder a bug is to find, the sillier the cause**.
* Once you find the problem, **create at least one test** to include in your test suite that exhibits the bug.

**Conclusions**

* The book [“Tao Te Programming”](http://www.burns-stat.com/documents/books/tao-te-programming/) by [Patrick Burns](http://www.burns-stat.com/) is an amazing source of wisdom.
* Packages [tidygraph](https://github.com/thomasp85/tidygraph) and [ggraph](https://github.com/thomasp85/ggraph) by [Thomas Lin Pedersen](https://www.data-imaginist.com/) offer consistent “tidy” tools for network analysis.
* **To learn to program one should learn to debug**.

[sessionInfo()](http://www.questionflow.org/2018/03/06/tao-of-tidygraph/#sessionInfo)

sessionInfo()

## R version 3.4.3 (2017-11-30)

## Platform: x86\_64-pc-linux-gnu (64-bit)

## Running under: Ubuntu 16.04.4 LTS

##

## Matrix products: default

## BLAS: /usr/lib/openblas-base/libblas.so.3

## LAPACK: /usr/lib/libopenblasp-r0.2.18.so

##

## locale:

## [1] LC\_CTYPE=ru\_UA.UTF-8 LC\_NUMERIC=C

## [3] LC\_TIME=ru\_UA.UTF-8 LC\_COLLATE=ru\_UA.UTF-8

## [5] LC\_MONETARY=ru\_UA.UTF-8 LC\_MESSAGES=ru\_UA.UTF-8

## [7] LC\_PAPER=ru\_UA.UTF-8 LC\_NAME=C

## [9] LC\_ADDRESS=C LC\_TELEPHONE=C

## [11] LC\_MEASUREMENT=ru\_UA.UTF-8 LC\_IDENTIFICATION=C

##

## attached base packages:

## [1] methods stats graphics grDevices utils datasets base

##

## other attached packages:

## [1] bindrcpp\_0.2 ggraph\_1.0.1 tidygraph\_1.1.0 forcats\_0.3.0

## [5] stringr\_1.3.0 dplyr\_0.7.4 purrr\_0.2.4 readr\_1.1.1

## [9] tidyr\_0.8.0.9000 tibble\_1.4.2 ggplot2\_2.2.1 tidyverse\_1.2.1

## [13] taoteprog\_0.0.1 igraph\_1.1.2

##

## loaded via a namespace (and not attached):

## [1] ggrepel\_0.7.0 Rcpp\_0.12.15 lubridate\_1.7.3

## [4] lattice\_0.20-35 utf8\_1.1.3 assertthat\_0.2.0

## [7] rprojroot\_1.3-2 digest\_0.6.15 psych\_1.7.8

## [10] ggforce\_0.1.1 R6\_2.2.2 cellranger\_1.1.0

## [13] plyr\_1.8.4 backports\_1.1.2 evaluate\_0.10.1

## [16] httr\_1.3.1 blogdown\_0.5 pillar\_1.2.1

## [19] rlang\_0.2.0 lazyeval\_0.2.1 readxl\_1.0.0

## [22] rstudioapi\_0.7 rmarkdown\_1.9 labeling\_0.3

## [25] udunits2\_0.13 foreign\_0.8-69 munsell\_0.4.3

## [28] broom\_0.4.3 compiler\_3.4.3 modelr\_0.1.1

## [31] xfun\_0.1 pkgconfig\_2.0.1 mnormt\_1.5-5

## [34] htmltools\_0.3.6 gridExtra\_2.3 bookdown\_0.7

## [37] viridisLite\_0.3.0 crayon\_1.3.4 MASS\_7.3-49

## [40] grid\_3.4.3 nlme\_3.1-131 jsonlite\_1.5

## [43] gtable\_0.2.0 magrittr\_1.5 units\_0.5-1

## [46] scales\_0.5.0 cli\_1.0.0 stringi\_1.1.6

## [49] reshape2\_1.4.3 viridis\_0.5.0 xml2\_1.2.0

## [52] tools\_3.4.3 glue\_1.2.0 tweenr\_0.1.5

## [55] hms\_0.4.1 parallel\_3.4.3 yaml\_2.1.17

## [58] colorspace\_1.3-2 rvest\_0.3.2 knitr\_1.20

## [61] bindr\_0.1 haven\_1.1.1