`criticalpath`: Método do Caminho Crítico em **R**

Rubens José Rosa, Marcos dos Santos, Thiago Marques

28/01/2022

Contents

1 Introdução **2**

2 Opções para criar um cronograma **3**

2.1 Introdução . . . 3

2.2 Alternativas . . . 4

2.3 Diagrama de rede . . . 6

3 Como criar um cronograma **8**

3.1 Criar o cronograma e adicionar cada atividade . . . 8

3.2 Criar o cronograma, adicionar cada atividade e cada relacionamento . . . 9

3.3 Criar o cronograma e adicionar cada atividade, ao mesmo tempo em que adiciona os relacionamentos dessa atividade com as outras . . . 10

3.4 Criar o cronograma, adicionar as atividades e os relacionamentos a partir de tibbles implícitos 12

3.5 Criar o cronograma, adicionar as atividades e os relacionamentos a partir de tibbles explícitos 13

3.6 Cronograma exemplo . . . 15

4 Como recuperar informações do cronograma **16**

5 Como recuperar as informações das atividades **17**

5.1 O cronograma possui alguma atividade? Quantas? . . . 17

5.2 Recuperando uma atividade ou todas as atividades . . . 18

5.3 Como mudar a duração das atividades . . . 20

5.4 Matriz de Gantt (Gráfico de Gantt) . . . 21

6 Como recuperar as informações dos relacionamentos **23**

6.1 O cronograma possui algum relacionamento? Quantos? . . . 24

6.2 Recuperando os relacionamentos . . . 24

6.3 Atividades predecessoras e sucessoras . . . 25

7 Como recupearar Indicadores topológicos **28**

8 Referências **30**

9 Autores do pacote `criticalpath` **31**

9.1 Rubens José Rosa 32

9.2 Marcos dos Santos 33

9.3 Thiago Marques 34

1 Introdução

O *Project Management Institue - PMI* é uma referência mundial em gerenciamento de projetos (mais informações no Site). Ele define projeto como sendo "um esforço temporário empreendido para criar um produto, serviço ou resultado único", e complementa afirmando que "sua natureza temporária indica um início e um término definidos" [32]. Pela definição de projeto e o complemento, o tempo de execução do projeto é fundamental para o seu sucesso.

Existem várias metodologias para calcular a duração do projeto, entre elas, o PMI cita o Método do Caminho Crítico - *Critical Path Method - CPM*. O CPM calcula a duração do projeto através da definição de suas atividades, a duração delas e a relação de precedência lógica (RPL) entre as mesmas.

O pacote `criticalpath` é uma implementação do método do caminho crítico em **R**. Ele segue os passos do algoritmo do CPM conforme definido no Guia PMBOK [32] e também no livro de Mario Vanhoucke [3], outra referência mundia em estudos de cronograma de projetos.

O **R** é um ambiente estatísticos utilizado para fazer análise de dados de vários tipos de experimentos. Uma das ferramentas do gerenciamento de projetos é o *Design of Experiment (DOE)*. Então, esse pacote foi criado para permitir que gerentes de projetos e pesquisadores possam fazer seus experimentos com o método do caminho crítico.

> O objetivo deste texto é mostrar como utilizar o pacote `criticalpath` para criar cronogramas seguindo o Método do Caminho Crítico em **R**.

Com este pacote e algumas funções do **R**, você pode calcular os seguintes parâmetros do CPM:

- duração do cronograma;
- data de início e término mais cedo;
- data de início e término mais tarde;
- caminho crítico;
- atividades críticas;
- folga total e folga livre;
- gráfico de Gantt;
- análise de cenários "E-se";
- indicadores topológicos;

O objetivo deste pacote é permitir que gerente de projetos e pesquisadores utilizem o método do caminho crítico em suas pesquisas para fazer experimentos com os parâmetros calculados pelo CPM, a fim de melhorar o desempenho dos projetos.

Este documento é totalmente voltado ao pacote `criticalpath`. Também serão utilizados outros pacotes. Então, já vamos carregar todos eles aqui.

```
library(criticalpath)
library(DiagrammeR)
library(DiagrammeRsvg)
#> Warning: package 'DiagrammeRsvg' was built under R version 4.1.2
library(png)
library(rsvg)
#> Warning: package 'rsvg' was built under R version 4.1.2
#> Linking to librsvg 2.48.8
library(tibble)
#> Warning: package 'tibble' was built under R version 4.1.2
```

Observação 1: O pacote faz extenso uso de **pipes**, representandos pelos símbolos **%>%** e **%<>%**, desta forma, para facilitar a sua utilização, ele importou os pipes do pacote `magrittr`. Então, quando você carregar o pacote `criticalpath`, ele já carrega os pipes.

Observação 2: Para instalar o pacote `criticalpath`, você tem duas opções:

- Versão estável (CRAN): `install.packages("criticalpath")`
- Versão em desenvolvimento (GitHub):
  - `install.packages("remotes")`
  - `remotes::install_github("rubens2005/criticalpath")`

2 Opções para criar um cronograma

2.1 Introdução

Para calcular o caminho crítico no **R**, a primeira coisa que precisamos saber é como representar um cronograma e como passar essa representação para o **R**.

Um cronograma é representado por uma lista de atividades e uma lista de relacionamentos entre elas. Ele é identificado por um título com uma referência da sua origem.

Para criar um cronograma, você deve fornecer um título, a referência sobre a sua origem (livro, artigo, empresa), a lista de atividades e a lista de relacionamentos. Após isso, é necessário planejar o cronograma, que nada mais é do que calcular o caminho crítico e todas as informações relativas ao projeto.

A criação de um cronograma requer uma sequência de passos e cada passo tem uma função **R** associada. Primeiro vamos ver os passos:

a) criar o cronograma;
b) identificar o cronograma;
c) adicionar as atividades;
d) adicionar os relacionamentos;
e) planejar o cronograma.

Os passos (a), (c) e (e) são obrigatórios. Não tem como criar um cronograma sem eles. No passo (e) é onde ocorre o cálculo do caminho crítico. Os passos (b) e (d) são opcionais, mas é altamente recomendado que você identifique seu cronograma e que os relacionamentos entre as atividades sejam definidos, senão todas atividades serão planejadas para serem executadas em paralelo.

Você tem cinco alternativas para criar um cronograma:

1. criar o cronograma e adicionar cada atividade;
2. criar o cronograma, adicionar cada atividade e cada relacionamento;
3. criar o cronograma e adicionar cada atividade, ao mesmo tempo em que adiciona os relacionamentos dessa atividade com as outras;
4. criar o cronograma, adicionar as atividades e os relacionamentos a partir de tibbles implícitos;
5. criar o cronograma, adicionar as atividades e os relacionamentos a partir de tibbles explícitos.

2.2 Alternativas

Vamos ver exemplos de códigos de cada uma dessas opções. Depois vamos descrever os detalhes delas.

1. Criar o cronograma e adicionar cada atividade:

```
# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades;
  sch_add_activity(1L, "A", 6L) %>%
  sch_add_activity(2L, "B", 5L) %>%
  sch_add_activity(3L, "C", 8L) %>%
  sch_add_activity(4L, "D", 9L) %>%
  sch_add_activity(5L, "E", 4L) %>%
  sch_add_activity(6L, "F", 3L) %>%
# e. planejar o cronograma.
  sch_plan()
```

2. Criar o cronograma, adicionar cada atividade e cada relacionamento:

```
# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades;
  sch_add_activity(1L, "A", 6L) %>%
  sch_add_activity(2L, "B", 5L) %>%
  sch_add_activity(3L, "C", 8L) %>%
  sch_add_activity(4L, "D", 9L) %>%
  sch_add_activity(5L, "E", 4L) %>%
  sch_add_activity(6L, "F", 3L) %>%
# d. adicionar os relacionamentos;
  sch_add_relation(1L, 2L) %>%
  sch_add_relation(1L, 3L) %>%
  sch_add_relation(2L, 4L) %>%
  sch_add_relation(3L, 4L) %>%
  sch_add_relation(3L, 5L) %>%
  sch_add_relation(4L, 6L) %>%
  sch_add_relation(5L, 6L) %>%
# e. planejar o cronograma.
  sch_plan()
```

3. Criar o cronograma e adicionar cada atividade, ao mesmo tempo em que adiciona os relacionamentos dessa atividade com as outras:

```
# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades; # d. adicionar os relacionamentos;
  sch_add_activity(1L, "A", 6L, 2L, 3L) %>%
  sch_add_activity(2L, "B", 5L, 4L) %>%
  sch_add_activity(3L, "C", 8L, 4L, 5L) %>%
  sch_add_activity(4L, "D", 9L, 6L) %>%
  sch_add_activity(5L, "E", 4L, 6L) %>%
  sch_add_activity(6L, "F", 3L) %>%
# e. planejar o cronograma.
  sch_plan()
```

4. Criar o cronograma, adicionar as atividades e os relacionamentos a partir de tibbles implícitos:

```
# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades;
  sch_add_activities(
    id = c(1L, 2L, 3L, 4L, 5L, 6L),
    name = c("A", "B", "C", "D", "E", "F"),
    duration = c(6L, 5L, 8L, 9L, 4L, 3L)
  ) %>%
# d. adicionar os relacionamentos;
  sch_add_relations(
    from = c(1L, 1L, 2L, 3L, 3L, 4L, 5L),
    to = c(2L, 3L, 4L, 4L, 5L, 6L, 6L)
  ) %>%
# e. planejar o cronograma.
  sch_plan()
```

5. Criar o cronograma, adicionar as atividades e os relacionamentos a partir de tibbles explícitos.

```
# Preparação: primeiro criar os tibbles
atb <- tibble(
  id = c(1L, 2L, 3L, 4L, 5L, 6L),
  name = c("A", "B", "C", "D", "E", "F"),
  duration = c(6L, 5L, 8L, 9L, 4L, 3L)
)
rtb <- tibble(
  from = c(1L, 1L, 2L, 3L, 3L, 4L, 5L),
  to = c(2L, 3L, 4L, 4L, 5L, 6L, 6L)
)
```

```
# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades;
  sch_add_activities_tibble(atb) %>%
# d. adicionar os relacionamentos;
  sch_add_relations_tibble(rtb) %>%
# e. planejar o cronograma.
  sch_plan()
```

2.3 Diagrama de rede

Todos os exemplos mostrados aqui são do mesmo cronograma. Foi feito assim para que você possa comparar as diversas formas de criar os cronogramas. Vamos ver como é o diagrama de rede desse cronograma. O código abaixo fica como exemplo para a criação do diagrama de rede de qualquer cronograma, basta chamar a função com o cronograma desejado.

```
criar_diagrama_de_rede <- function(sch) {
  atb <- sch_activities(sch)
  rtb <- sch_relations(sch)
  ids <- atb$id

  ndf <-
    create_node_df(
      n         = sch_nr_activities(sch),
      label     = ids,
      color     = "black",
      fontcolor = "black",
      shape     = ifelse(atb$milestone, "diamond","rect"),
      fillcolor = ifelse(atb$critical, "#ff7555", "#6399ea"),
    )


  edf <-
    create_edge_df(
      from     = match(rtb$from, ids),
      to       = match(rtb$to, ids),
      rel      = rtb$type,
      penwidth = ifelse(rtb$critical, "3", "1"),
      color    = ifelse(rtb$critical, "#ff7555", "#6399ea"),
      style    = "solid"
    )

  graph <-
    create_graph(
      nodes_df = ndf,
      edges_df = edf
    ) %>%
    add_global_graph_attrs(
      attr = "layout",
```

```
      value = "dot",
      attr_type = "graph"
    ) %>%
    add_global_graph_attrs(
      attr = "rankdir",
      value = "LR",
      attr_type = "graph"
    ) %>%
    set_node_attrs(
      node_attr = "fontname",
      values = "Helvetica"
    ) %>%
    set_node_attrs(
      node_attr = "fontsize",
      values = "20"
    )

  graph
}

gerar_arquivo_do_diagrama <- function(sch) {
  # Criar um arquivo temporário
  out <- paste0(tempfile(), ".png")

  # Gerar o diagrama dem PNG.
  svg <- sch %>%
    criar_diagrama_de_rede() %>%
    generate_dot() %>%
    grViz() %>%
    export_svg() %>%
    charToRaw() %>%
    rsvg() %>%
    writePNG(out)

  return(out)
}

diagrama_original <- sch %>%
  gerar_arquivo_do_diagrama()
```

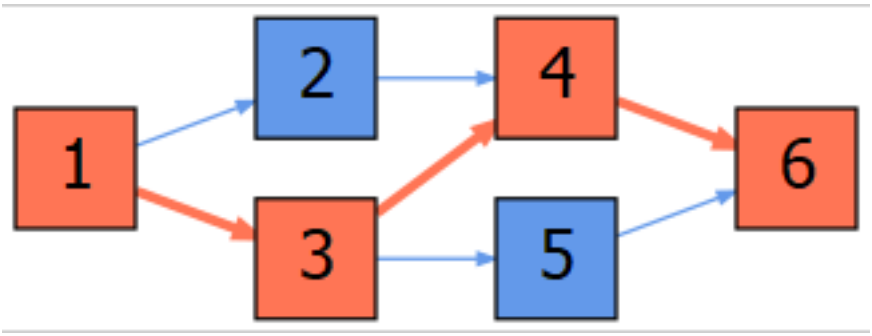


Figure 1: Diagrama de rede

O diagrama de rede mostra a Relação de Precedência Lógicas (RPL) entre as atividades. Por exemplo, para poder executar a atividade 4, é necessário finalizar antes as atividades 2 e 3, que depende do término da 1. Essa mesma lógica serve para todas as atividades.

Em vermelho, está o caminho crítico, formado pelas atividades 1, 3, 4 e 6. A soma da duração dessas atividades é que determina a duração do projeto. Nenhum outro caminho dura mais que o crítico. Na verdade, essa é a definição de caminho crítico: é o caminho com a maior duração.

Esse diagrama foi criado a partir das definições do cronograma feitas pelo pacote `criticalpath`. Vamos ver, então, como criar um cronograma.

3 Como criar um cronograma

Agora vamos discutir cada uma das alternativas para criar o nosso cronograma.

3.1 Criar o cronograma e adicionar cada atividade

- **Add Activity:** Adiciona uma atividade no cronograma.
  - Utilização:
    - `sch_add_activity(sch, id, name, duration)`
      - **id:** Identificador da atividade. É um número inteiro que deve ser único: não pode haver duas atividades como o mesmo identificador. Esse identificador será utilizado para fazer o relacionamento entre as atividades.
      - **name:** Nome da atividade.
      - **duration:** Duração da atividade. Seu formato deve ser um número inteiro. Ele é um número independente de unidade de tempo. O pacote `criticalpath` interpreta como período, que pode ser dias, horas, minutos ou segundos. Deve ser um valor maior ou igual a zero.

Na seção Criar o cronograma e adicionar cada atividade, ao mesmo tempo em que adiciona os relacionamentos dessa atividade com as outras será mostrado a mesma função, mas com parâmetros adicionais.

Exemplos:

```
# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades;
  sch_add_activity(1L, "A", 6L) %>%
  sch_add_activity(2L, "B", 5L) %>%
  sch_add_activity(3L, "C", 8L) %>%
  sch_add_activity(4L, "D", 9L) %>%
  sch_add_activity(5L, "E", 4L) %>%
  sch_add_activity(6L, "F", 3L) %>%
# e. planejar o cronograma.
  sch_plan()

sch_duration(sch)
#> [1] 9
sch_activities(sch)
#> # A tibble: 6 x 14
#>      id name  duration milestone critical early_start early_finish late_start
#>   <int> <chr>    <int> <lgl>     <lgl>          <int>        <int>      <int>
#> 1     1 A            6 FALSE     FALSE              0            6          3
#> 2     2 B            5 FALSE     FALSE              0            5          4
#> 3     3 C            8 FALSE     FALSE              0            8          1
#> 4     4 D            9 FALSE     TRUE               0            9          0
#> 5     5 E            4 FALSE     FALSE              0            4          5
#> 6     6 F            3 FALSE     FALSE              0            3          6
#> # ... with 6 more variables: late_finish <int>, total_float <int>,
#> #   free_float <int>, progr_level <int>, regr_level <int>, topo_float <int>
```

3.2 Criar o cronograma, adicionar cada atividade e cada relacionamento

Um cronograma é formado por atividades e o relacionamento entre elas. Neste tópico vamos ver como fazemos o relacionamento entre as atividades.

- **Add Relation:** Adiciona no cronograma um relacionamento entre duas atividades.
- Utilização:
  - `sch_add_relation(from, to, type="FS", lag=0)`
    - **from:** Identificador da atividade predecessora. Deve existir uma atividade com esse identificador.
    - **to:** Identificador da atividade sucessora. Deve existir uma atividade com esse identificador.
    - **type:** Tipo do relacionamento entre as atividades. Seus valores podem ser: FS, FF, SS, SF. O relacionamento padrão é o FS.
      - `FS`: Término-Início: A atividade sucessora só pode começar depois que a predecessora finalizar.
      - `FF`: Término-Término: A atividade sucessora deve finalizar junto com a predecessora.
      - `SS`: Início-Início: A atividade sucessora deve iniciar junto com a predecessora.
      - `SF`: Início-Término: A atividade sucessora deve finalizar quando ao predecessora iniciar.
    - **lag:** Período de tempo que a atividade sucessora deve esperar para ser executada. Por exemplo, num relacionamento FS, com `lag = 4`, após a predecessora finalizar, a sucessora deve esperar 4 períodos. Em outras palavras, a atividade sucessora avança 4 períodos após

o término da predecessora. Se o valor for negativo, a atividade sucessora antecipa seu início. No exemplo acima, se `lag = -4`, a atividade sucessora adiantaria em 4 períodos o seu início. Por padrão, `lag = 0`.

Exemplos:

```
# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades;
  sch_add_activity(1L, "A", 6L) %>%
  sch_add_activity(2L, "B", 5L) %>%
  sch_add_activity(3L, "C", 8L) %>%
  sch_add_activity(4L, "D", 9L) %>%
  sch_add_activity(5L, "E", 4L) %>%
  sch_add_activity(6L, "F", 3L) %>%
# d. adicionar os relacionamentos;
  sch_add_relation(1L, 2L) %>%
  sch_add_relation(1L, 3L) %>%
  sch_add_relation(2L, 4L) %>%
  sch_add_relation(3L, 4L) %>%
  sch_add_relation(3L, 5L) %>%
  sch_add_relation(4L, 6L) %>%
  sch_add_relation(5L, 6L) %>%
# e. planejar o cronograma.
  sch_plan()

sch_duration(sch)
#> [1] 26
sch_relations(sch)
#> # A tibble: 7 x 8
#>    from    to type    lag critical   ord i_from  i_to
#>   <int> <int> <chr> <int> <lgl>    <int>  <int> <int>
#> 1     1     2 FS        0 FALSE        1      1     2
#> 2     1     3 FS        0 TRUE         2      1     3
#> 3     2     4 FS        0 FALSE        3      2     4
#> 4     3     4 FS        0 TRUE         4      3     4
#> 5     3     5 FS        0 FALSE        5      3     5
#> 6     4     6 FS        0 TRUE         6      4     6
#> 7     5     6 FS        0 FALSE        7      5     6
```

3.3 Criar o cronograma e adicionar cada atividade, ao mesmo tempo em que adiciona os relacionamentos dessa atividade com as outras

- **Add Activity:** Adiciona uma atividade no cronograma. Tem também a opção de já adicionar os relacionamentos que a atividade possui.
- Utilização:
  - `sch_add_activity(sch, id, name, duration, ..., direction = "succ")`
    - **id:** Identificador da atividade. É um número inteiro que deve ser único: não pode haver duas atividades como o mesmo identificador. Esse identificador será utilizado para fazer o relacionamento entre as atividades.

- **name:** Nome da atividade.
- **duration:** Duração da atividade. Seu formato deve ser um número inteiro. Ele é um número independente de unidade de tempo. O pacote `criticalpath` interpreta como período, que pode ser dias, horas, minutos ou segundos. Deve ser um valor maior ou igual a zero.
- "...": Um conjunto de identificadores das atividades que terão um relacionamento com a nova atividade.
- **direction:** Indica a direção do relacionamento que a atividade terá com os outros indentificadores informados:
  - `succ`: o conjunto de identificadores serão sucessoras da nova atividade;
  - `pred`: o conjunto de identifidadores serão predecessoras da nova atividade.

A função descrita aqui é a mesma de Criar o cronograma e adicionar cada atividade. A diferença é que aqui foram mostrados mais parâmetros.

Exemplos:

```
# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades; # d. adicionar os relacionamentos;
  sch_add_activity(1L, "A", 6L, 2L, 3L) %>%
  sch_add_activity(2L, "B", 5L, 4L) %>%
  sch_add_activity(3L, "C", 8L, 4L, 5L) %>%
  sch_add_activity(4L, "D", 9L, 6L) %>%
  sch_add_activity(5L, "E", 4L, 6L) %>%
  sch_add_activity(6L, "F", 3L) %>%
# e. planejar o cronograma.
  sch_plan()

sch_duration(sch)
#> [1] 26
sch_activities(sch)
#> # A tibble: 6 x 14
#>      id name  duration milestone critical early_start early_finish late_start
#>   <int> <chr>    <int> <lgl>     <lgl>          <int>        <int>      <int>
#> 1     1 A            6 FALSE     TRUE               0            6          0
#> 2     2 B            5 FALSE     FALSE              6           11          9
#> 3     3 C            8 FALSE     TRUE               6           14          6
#> 4     4 D            9 FALSE     TRUE              14           23         14
#> 5     5 E            4 FALSE     FALSE             14           18         19
#> 6     6 F            3 FALSE     TRUE              23           26         23
#> # ... with 6 more variables: late_finish <int>, total_float <int>,
#> #   free_float <int>, progr_level <int>, regr_level <int>, topo_float <int>
sch_relations(sch)
#> # A tibble: 7 x 8
#>    from    to type    lag critical   ord i_from  i_to
#>   <int> <int> <chr> <int> <lgl>    <int>  <int> <int>
#> 1     1     2 FS        0 FALSE        1      1     2
#> 2     1     3 FS        0 TRUE         2      1     3
#> 3     2     4 FS        0 FALSE        3      2     4
#> 4     3     4 FS        0 TRUE         4      3     4
#> 5     3     5 FS        0 FALSE        5      3     5
```

```
#> 6     4     6 FS        0 TRUE        6     4     6
#> 7     5     6 FS        0 FALSE       7     5     6
```

3.4 Criar o cronograma, adicionar as atividades e os relacionamentos a partir de tibbles implícitos

Tanto as atividades como os relacionamentos podem ser adicionados ao cronograma a partir de tibbles implícitos. Eles são implícitos, pois a criação dos tibbles é feita por uma função, não diretamente por nós.

- **Add Activities:** Combina vários vetores com as informações sobre as atividades num tibble para adicioná-las ao cronograma.
- Utilização:
  - `sch_add_activities(sch, id, name, duration)`
    - **id:** Identificador da atividade. É um número inteiro que deve ser único: não pode haver duas atividades como o mesmo identificador. Esse identificador será utilizado para fazer o relacionamento entre as atividades.
    - **name:** Nome da atividade.
    - **duration:** Duração da atividade. Seu formato deve ser um número inteiro. Ele é um número independente de unidade de tempo. O pacote `criticalpath` interpreta como período, que pode ser dias, horas, minutos ou segundos. Deve ser um valor maior ou igual a zero.
- **Add Relations:** Combina vários vetores num tibble para criar o relacionamento entre as atividades e os adiciona ao cronograma.
- Utilização:
  - `sch_add_relations(from, to, type="FS", lag = 0)`
    - **from:** Vetor de identificadores da atividade predecessora. Deve existir uma atividade com esse identificador.
    - **to:** Identificador da atividade sucessora. Deve existir uma atividade com esse identificador.
    - **type:** Tipo do relacionamento entre as atividades. Seus valores podem ser: FS, FF, SS, SF. O relacionamento padrão é o FS.
      - `FS`: Término-Início: A atividade sucessora só pode começar depois que a predecessora finalizar.
      - `FF`: Término-Término: A atividade sucessora deve finalizar junto com a predecessora.
      - `SS`: Início-Início: A atividade sucessora deve iniciar junto com a predecessora.
      - `SF`: Início-Término: A atividade sucessora deve finalizar quando ao predecessora iniciar.
    - **lag:** Período de tempo que a atividade sucessora deve esperar para ser executada. Por exemplo, num relacionamento FS, com `lag = 4`, após a predecessora finalizar, a sucessora deve esperar 4 períodos. Em outras palavras, a atividade sucessora avança 4 períodos após o término da predecessora. Se o valor for negativo, a atividade sucessora antecipa seu início. No exemplo acima, se `lag = -4`, a atividade sucessora adiantaria em 4 períodos o seu início. Por padrão, `lag = 0`.

Exemplos:

```
# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades;
```

```
  sch_add_activities(
    id = c(1L, 2L, 3L, 4L, 5L, 6L),
    name = c("A", "B", "C", "D", "E", "F"),
    duration = c(6L, 5L, 8L, 9L, 4L, 3L)
  ) %>%
# d. adicionar os relacionamentos;
  sch_add_relations(
    from = c(1L, 1L, 2L, 3L, 3L, 4L, 5L),
    to = c(2L, 3L, 4L, 4L, 5L, 6L, 6L)
  ) %>%
# e. planejar o cronograma.
  sch_plan()

sch_duration(sch)
#> [1] 26
sch_activities(sch)
#> # A tibble: 6 x 14
#>      id name  duration milestone critical early_start early_finish late_start
#>   <int> <chr>    <int> <lgl>     <lgl>          <int>        <int>      <int>
#> 1     1 A            6 FALSE     TRUE               0            6          0
#> 2     2 B            5 FALSE     FALSE              6           11          9
#> 3     3 C            8 FALSE     TRUE               6           14          6
#> 4     4 D            9 FALSE     TRUE              14           23         14
#> 5     5 E            4 FALSE     FALSE             14           18         19
#> 6     6 F            3 FALSE     TRUE              23           26         23
#> # ... with 6 more variables: late_finish <int>, total_float <int>,
#> #   free_float <int>, progr_level <int>, regr_level <int>, topo_float <int>
sch_relations(sch)
#> # A tibble: 7 x 8
#>    from    to type    lag critical   ord i_from  i_to
#>   <int> <int> <chr> <int> <lgl>    <int>  <int> <int>
#> 1     1     2 FS        0 FALSE        1      1     2
#> 2     1     3 FS        0 TRUE         2      1     3
#> 3     2     4 FS        0 FALSE        3      2     4
#> 4     3     4 FS        0 TRUE         4      3     4
#> 5     3     5 FS        0 FALSE        5      3     5
#> 6     4     6 FS        0 TRUE         6      4     6
#> 7     5     6 FS        0 FALSE        7      5     6
```

3.5 Criar o cronograma, adicionar as atividades e os relacionamentos a partir de tibbles explícitos

Tanto as atividades como os relacionamentos podem ser adicionados ao cronograma a partir de tibbles explícitos. Eles são explícitos, pois nós mesmo os criamos, antes de chamar a função responsáveis por adicioná-los ao cronograma.

- **Add Activities Tibble:** Adiciona um tibble com as informações das atividades. A estrutura do tibble está descrita logo abaixo.
- Utilização:
  - `sch_add_activities_tibble(sch, atb)`
    - **id:** Identificador da atividade. É um número inteiro que deve ser único: não pode haver duas atividades como o mesmo identificador. Esse identificador será utilizado para fazer o

relacionamento entre as atividades.
    * **name:** Nome da atividade.
    * **duration:** Duração da atividade. Seu formato deve ser um número inteiro. Ele é um número independente de unidade de tempo. O pacote `criticalpath` interpreta como período, que pode ser dias, horas, minutos ou segundos. Deve ser um valor maior ou igual a zero.

- **Add Relations:** Adiciona um tibble com as informações dos relacionamentos entre as atividades. A estrutura do tibble está descrita logo abaixo.
- Utilização:
  - `sch_add_relations(from, to, type="FS", lag = 0)`
    * **from:** Vetor de identificadores da atividade predecessora. Deve existir uma atividade com esse identificador.
    * **to:** Identificador da atividade sucessora. Deve existir uma atividade com esse identificador.
    * **type:** Tipo do relacionamento entre as atividades. Seus valores podem ser: FS, FF, SS, SF. O relacionamento padrão é o FS.
      · `FS`: Término-Início: A atividade sucessora só pode começar depois que a predecessora finalizar.
      · `FF`: Término-Término: A atividade sucessora deve finalizar junto com a predecessora.
      · `SS`: Início-Início: A atividade sucessora deve iniciar junto com a predecessora.
      · `SF`: Início-Término: A atividade sucessora deve finalizar quando ao predecessora iniciar.
    * **lag:** Período de tempo que a atividade sucessora deve esperar para ser executada. Por exemplo, num relacionamento FS, com `lag = 4`, após a predecessora finalizar, a sucessora deve esperar 4 períodos. Em outras palavras, a atividade sucessora avança 4 períodos após o término da predecessora. Se o valor for negativo, a atividade sucessora antecipa seu início. No exemplo acima, se `lag = -4`, a atividade sucessora adiantaria em 4 períodos o seu início. Por padrão, `lag = 0`.

Exemplos:

```
# Preparação: primeiro criar os tibbles
atb <- tibble(
  id = c(1L, 2L, 3L, 4L, 5L, 6L),
  name = c("A", "B", "C", "D", "E", "F"),
  duration = c(6L, 5L, 8L, 9L, 4L, 3L)
)
rtb <- tibble(
  from = c(1L, 1L, 2L, 3L, 3L, 4L, 5L),
  to = c(2L, 3L, 4L, 4L, 5L, 6L, 6L)
)

# a. criar o cronograma;
sch <- sch_new() %>%
# b. identificar o cronograma;
  sch_title("Cronograma Exemplo") %>%
  sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
# c. adicionar as atividades;
  sch_add_activities_tibble(atb) %>%
# d. adicionar os relacionamentos;
  sch_add_relations_tibble(rtb) %>%
# e. planejar o cronograma.
  sch_plan()

sch_duration(sch)
```

```
#> [1] 26
sch_activities(sch)
#> # A tibble: 6 x 14
#>      id name  duration milestone critical early_start early_finish late_start
#>   <int> <chr>    <int> <lgl>     <lgl>          <int>        <int>      <int>
#> 1     1 A            6 FALSE     TRUE               0            6          0
#> 2     2 B            5 FALSE     FALSE              6           11          9
#> 3     3 C            8 FALSE     TRUE               6           14          6
#> 4     4 D            9 FALSE     TRUE              14           23         14
#> 5     5 E            4 FALSE     FALSE             14           18         19
#> 6     6 F            3 FALSE     TRUE              23           26         23
#> # ... with 6 more variables: late_finish <int>, total_float <int>,
#> #   free_float <int>, progr_level <int>, regr_level <int>, topo_float <int>
sch_relations(sch)
#> # A tibble: 7 x 8
#>    from    to type    lag critical   ord i_from  i_to
#>   <int> <int> <chr> <int> <lgl>    <int>  <int> <int>
#> 1     1     2 FS        0 FALSE        1      1     2
#> 2     1     3 FS        0 TRUE         2      1     3
#> 3     2     4 FS        0 FALSE        3      2     4
#> 4     3     4 FS        0 TRUE         4      3     4
#> 5     3     5 FS        0 FALSE        5      3     5
#> 6     4     6 FS        0 TRUE         6      4     6
#> 7     5     6 FS        0 FALSE        7      5     6
```

3.6 Cronograma exemplo

Como vamos fazer diversos testes neste documento, vou criar uma função para criar / recriar o nosso cronograma exemplo. Também vou criar uma função para retornar um cronograma vazio, sem atividades e sem relacionamentos. É claro que nesse cronograma o caminho crítico não pode ser calculado.

```
cronograma_exemplo <- function() {
  out <- sch_new() %>%
    sch_title("Cronograma Exemplo") %>%
    sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>") %>%
    sch_add_activities(
      id = c(1L, 2L, 3L, 4L, 5L, 6L),
      name = c("A", "B", "C", "D", "E", "F"),
      duration = c(6L, 5L, 8L, 9L, 4L, 3L)
    ) %>%
    sch_add_relations(
      from = c(1L, 1L, 2L, 3L, 3L, 4L, 5L),
      to = c(2L, 3L, 4L, 4L, 5L, 6L, 6L)
    ) %>%
    sch_plan()
  return(out)
}

cronograma_vazio <- function() {
  out <- sch_new() %>%
    sch_title("Cronograma Exemplo") %>%
    sch_reference("ROSA, Rubens José. (2022) <https://rubensjoserosa.com>")
```

```
  return(out)
}
```

4 Como recuperar informações do cronograma

Após a criação de um cronograma, é possível recuperar várias informações dele, como o título, a referência e a duração. Segue a descrição de cada uma delas.

- **title:** O título ou nome do projeto que está sob análise. Esse título depende do usuário da classe, pois ele deve identificar um projeto real ou algum experimento do próprio usuário.
- Utilização:
  - `sch_title(sch, "O Título")`: altera o título do projeto.
  - `sch_title(sch)`: recupera o título do projeto.
- **reference:** A referência da origem do projeto, por exemplo, um livro, um artigo, uma corporação ou, se não puder ser identificada, não precisa colocar nada.
- Utilização:
  - `sch_reference(sch, "Uma nova referência.")`: altera a referência do projeto.
  - `sch_reference(sch)`: recupera a referência do projeto.
- **duration:** Um valor inteiro que indica a duração do projeto calculada pelo CPM. A duração e todos os outros valores do método do caminho crítico são calculados após a chamada à função `sch_plan(sch)`, por isso, ela não pode ser alterada.
- Utilização:
  - `sch_duration(sch)`: recupera a duração do projeto.

Exemplos:

Consultando e alterando o título do cronograma

```
sch <- cronograma_exemplo()

sch_title(sch)
#> [1] "Cronograma Exemplo"

sch %<>%
  sch_title("Novo título")
sch_title(sch)
#> [1] "Novo título"
```

Consultando e alterando a referência do cronograma

```
sch <- cronograma_exemplo()

sch_reference(sch)
#> [1] "ROSA, Rubens José. (2022) <https://rubensjoserosa.com>"

sch %<>%
  sch_reference("Nova referência (2022).")
sch_reference(sch)
#> [1] "Nova referência (2022)."
```

Consultando e "alterando" a duração do cronograma

```
sch <- cronograma_exemplo()

sch_duration(sch)
#> [1] 26

# A duração do cronograma só é alterada quando inclui uma nova atividade nele
# Vou adicionar a atividade 7L, tendo a 6L como predecessora.
sch %<>%
  sch_add_activity(7L, "G", 10L, 6L, direction = "pred") %>%
  sch_plan()

sch_duration(sch)
#> [1] 36

# Vou adicionar a atividade 8L, tendo a 1L como sucessora.
sch %<>%
  sch_add_activity(8L, "H", 10L, 1L) %>%
  sch_plan()

sch_duration(sch)
#> [1] 46
```

5 Como recuperar as informações das atividades

O cronograma possui diversas informações sobre as atividades. Após incluir as atividades, você precisa saber quantas foram incluídas, quando cada uma delas está planejada para iniciar ou finalizar. Esta seção explica como você pode recuperar essas e outras informações.

5.1 O cronograma possui alguma atividade? Quantas?

- **Has Any Activity:** Valor lógico que indica se o cronograma possui ou não alguma atividade.
  - `TRUE`: indica que o cronograma tem pelo menos uma atividade;
  - `FALSE`: indica que o cronograma não possui nenhuma atividade.
- Utilização:
  - `sch_has_any_activity(sch)`
- **Number of Activities:** Indica o número de atividades que o cronograma possui.
- Utilização:
  - `sch_nr_activities(sch)`

Exemplos:

```
sch <- cronograma_vazio()

sch_has_any_activity(sch)  # FALSE
#> [1] FALSE
sch_nr_activities(sch)     # 0
```

```
#> [1] 0

sch <- cronograma_exemplo()

sch_has_any_activity(sch)  # TRUE
#> [1] TRUE
sch_nr_activities(sch)     # 6
#> [1] 6
```

5.2 Recuperando uma atividade ou todas as atividades

As atividades são colocadas numa tabela, que no **R** é chamada de data-frame. Um tipo especial de data-frame são os tibbles. As atividades e os relacionamentos entre elas estão nos tibbles.

Cada propriedade das atividades é colocada numa coluna do tibble. Segue as funções para recuperar as informações das atividades e a descrição de cada uma delas.

- **Get activity:** Recupera uma atividade de acordo com o seu identificador. A estrutura retornada será explicada no próximo tópico.
- Utilização:
  - `sch_get_activity(sch, id)`
  - **id**: Identificador da atividade a ser recuperada.
- **Activities:** Retorna um tibble com todas as atividades do cronograma, na ordem em que elas foram incluídas. Estas são as principais informações calculadas pelo `criticalpath`.
- Utilização:
  - `sch_get_activities(sch)`
  - O tibble possui a seguinte estrutura:
    - **id:** Identificador da atividade. É um número inteiro que deve ser único em todo o cronograma: não pode haver duas atividades como o mesmo identificador.Esse identificador será utilizado para fazer o relacionamento entre as atividades. É obrigatório informar um identificador para a atividade.
    - **name:** Nome da atividade.
    - **duration:** Duração da atividade. Seu formato deve ser um número inteiro, maior ou igual a zero. Ele é um número independente de unidade de tempo. O pacote `criticalpath` interpreta como período, que pode ser dias, horas, minutos ou segundos.
    - **milestone:** Indica se a atividade é um marco ou não. Um marco (*milestone*) é uma atividade com duração zero.
      - `TRUE`: indica que é um marco;
      - `FALSE`: indica que não é um marco.
    - **critical:** Indica se a atividade é crítica ou não. Uma atividade é crítica se sua folga total é menor ou igual a zero.
      - `TRUE` indica que a atividade é crítica;
      - `FALSE` indica que a atividade não é crítica.
    - **early_start:** Início mais cedo: período de início mais cedo que uma atividade pode iniciar com a condição de que todas as suas predecessoras já tenham finalizadas.
    - **early_finish:** Término mais cedo: é o início mais cedo mais a duração da atividade.
    - **late_start:** Início mais tarde: é o término mais tarde da atividade menos a sua duração.
    - **late_finish:** Término mais tarde: período de término mais tarde que uma atividade pode finalizar com a condição de que não se sobreponha às suas sucessoras.

* **total_float:** Folga total: Quantidade de períodos que uma atividade pode atrasar sem aumentar a duração do projeto.
* **free_float:** Folga livre: Quantidade de períodos que uma atividade pode atrasar sem atrasar nenhuma atividade sucessora dela.
* **progr_level:** Nível progressivo: Ordem em que a atividade aparece no diagrama de rede, contada a partir das primeiras atividades. As atividades que não possuem predecessoras tem nível progressivel igual a um. O nível das outras atividades é o maior nível entre as suas predecessoras, mais um. O maior nível das atividades que não possuem sucessoras é chamada de `max_level`.
* **regr_level:** Nível regressivo: Ordem em que as atividades aparecem no digrama de rede, contada a partir das últimas atividades. O nível regressivo das atividades que não possuem sucessoras é igual ao `max_level`. O nível das outras atividades é menor valor do nível regressivo de suas sucessoras, menos um.
* **topo_float:** Folga topológica: Diferença entre o nível regressivo e o nível progressivo.

Exemplos:

Recuperando uma ou todas atividades

```
sch <- cronograma_exemplo()

sch_get_activity(sch, 1L) %>%
  str()
#> tibble [1 x 14] (S3: tbl_df/tbl/data.frame)
#>  $ id          : int 1
#>  $ name        : chr "A"
#>  $ duration    : int 6
#>  $ milestone   : logi FALSE
#>  $ critical    : logi TRUE
#>  $ early_start : int 0
#>  $ early_finish: int 6
#>  $ late_start  : int 0
#>  $ late_finish : int 6
#>  $ total_float : int 0
#>  $ free_float  : int 0
#>  $ progr_level : int 1
#>  $ regr_level  : int 1
#>  $ topo_float  : int 0

sch_get_activity(sch, 5L) %>%
  str()
#> tibble [1 x 14] (S3: tbl_df/tbl/data.frame)
#>  $ id          : int 5
#>  $ name        : chr "E"
#>  $ duration    : int 4
#>  $ milestone   : logi FALSE
#>  $ critical    : logi FALSE
#>  $ early_start : int 14
#>  $ early_finish: int 18
#>  $ late_start  : int 19
#>  $ late_finish : int 23
#>  $ total_float : int 5
#>  $ free_float  : int 5
#>  $ progr_level : int 3
```

```
#>  $ regr_level  : int 3
#>  $ topo_float  : int 0

sch_activities(sch) %>%
  str()
#> tibble [6 x 14] (S3: tbl_df/tbl/data.frame)
#>  $ id          : int [1:6] 1 2 3 4 5 6
#>  $ name        : chr [1:6] "A" "B" "C" "D" ...
#>  $ duration    : int [1:6] 6 5 8 9 4 3
#>  $ milestone   : logi [1:6] FALSE FALSE FALSE FALSE FALSE FALSE
#>  $ critical    : logi [1:6] TRUE FALSE TRUE TRUE FALSE TRUE
#>  $ early_start : int [1:6] 0 6 6 14 14 23
#>  $ early_finish: int [1:6] 6 11 14 23 18 26
#>  $ late_start  : int [1:6] 0 9 6 14 19 23
#>  $ late_finish : int [1:6] 6 14 14 23 23 26
#>  $ total_float : int [1:6] 0 3 0 0 5 0
#>  $ free_float  : int [1:6] 0 3 0 0 5 0
#>  $ progr_level : int [1:6] 1 2 2 3 3 4
#>  $ regr_level  : int [1:6] 1 2 2 3 3 4
#>  $ topo_float  : int [1:6] 0 0 0 0 0 0
```

5.3 Como mudar a duração das atividades

Uma das ferramentas e técnicas do gerenciamento de cronogramas é a análise "E-se?". Essa funcionalidade do pacote `criticalpath` permite fazer justamente a análise "E-se?" através da alteração da duração de todas as atividades ao mesmo. A forma implementada aqui é mais rápida do que criar todo o cronograma novamente.

Atenção: O vetor com as novas durações das atividades deve estar ordenados pela ordem de inclusão das atividades.

- Utilização:
  - `sch_change_durations(sch, new_durations)`

```
sch <- cronograma_exemplo()

#Project duration
sch_duration(sch) # 26
#> [1] 26

#Activities duration
sch_activities(sch)$duration
#> [1] 6 5 8 9 4 3

new_durations <- c(1L,2L,5L, 4L,3L, 2L)
sch %<>%
  sch_change_activities_duration(new_durations)

#Project duration
sch_duration(sch) # 12
#> [1] 12
```

```
#Activities duration
sch_activities(sch)$duration
#> [1] 1 2 5 4 3 2
```

5.4 Matriz de Gantt (Gráfico de Gantt)

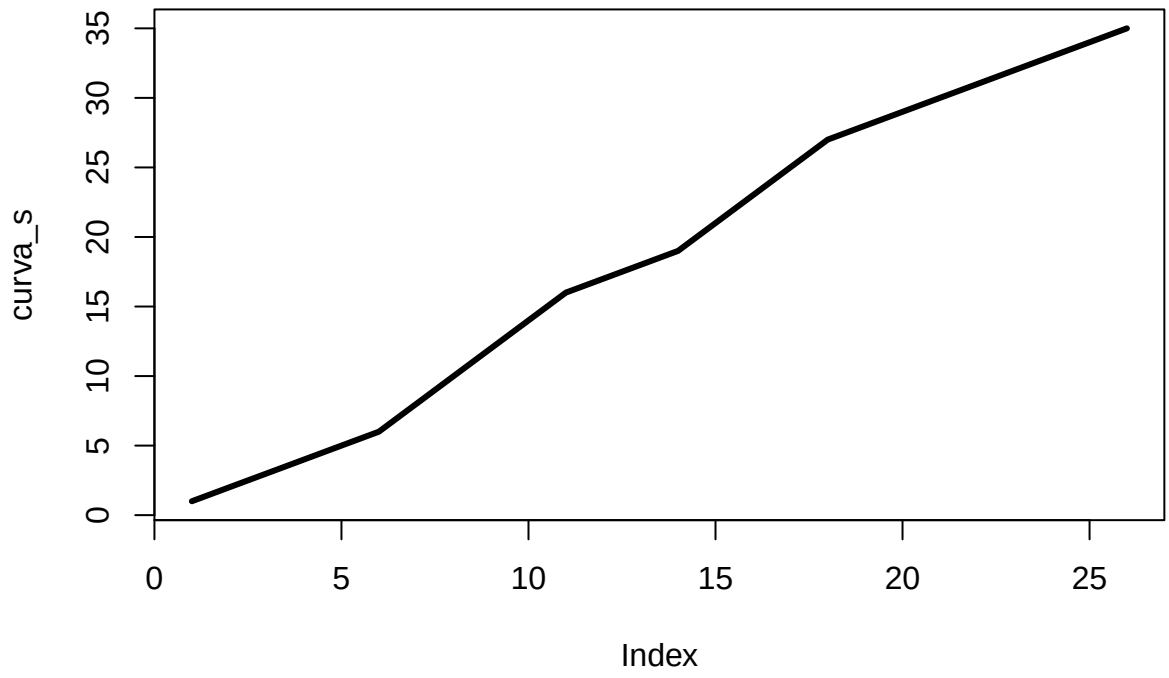
Cria uma matriz que representa o gráfico de Gantt do projeto. Na matriz, Um `1` indica que a atividade está planejada para ser executada no período. Zero `0` indica que ela não está planejada.

As linhas representam atividades e as colunas o período de execução do projeto. Então, o número de linhas é igual à quantidade de atividades e o número de colunas é igual à duração do projeto.

- Utilização:
  - `sch_gantt_matrix(sch)`: Retorna a matriz de Gantt.
  - `sch_xy_gantt_matrix(gantt)`: Transforma a matriz de Gantt em coordenadas ordenadas (x, y) e peso um. Cada ponto maior que zero na matriz torna-se uma coordenada (x, y).

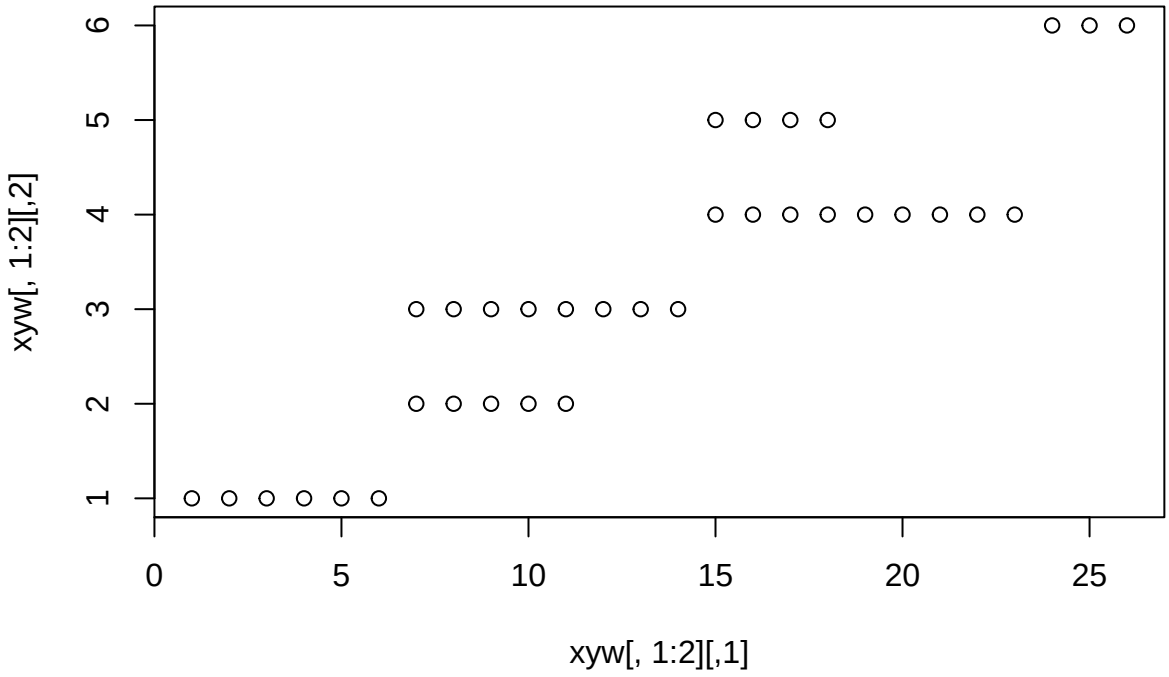
Exemplos:

```
sch <- cronograma_exemplo()

gantt <- sch_gantt_matrix(sch)
gantt
#>   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
#> 1 1 1 1 1 1 1 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
#> 2 0 0 0 0 0 0 1 1 1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
#> 3 0 0 0 0 0 0 1 1 1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0
#> 4 0 0 0 0 0 0 0 0 0  0  0  0  0  0  1  1  1  1  1  1  1  1  0  0  0
#> 5 0 0 0 0 0 0 0 0 0  0  0  0  0  0  1  1  1  1  0  0  0  0  0  0  0
#> 6 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1
#> attr(,"class")
#> [1] "Gantt"  "matrix" "array"

# Qual é o esforço total no período?
colSums(gantt)
#>  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
#>  1  1  1  1  1  1  2  2  2  2  2  1  1  1  2  2  2  2  1  1  1  1  1  1  1  1

# Qual é a duração de cada uma das atividades?
rowSums(gantt)
#> 1 2 3 4 5 6
#> 6 5 8 9 4 3

# Como criar uma Curva S?
curva_s <- cumsum(colSums(gantt))
curva_s
#>  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
#>  1  2  3  4  5  6  8 10 12 14 16 17 18 19 21 23 25 27 28 29 30 31 32 33 34 35
plot(curva_s, type="l", lwd=3)
```




```
xyw <- sch_xy_gantt_matrix(sch)
xyw
#>       [,1] [,2] [,3]
#>  [1,]    1    1    1
#>  [2,]    2    1    1
#>  [3,]    3    1    1
#>  [4,]    4    1    1
#>  [5,]    5    1    1
#>  [6,]    6    1    1
#>  [7,]    7    2    1
#>  [8,]    8    2    1
#>  [9,]    9    2    1
#> [10,]   10    2    1
#> [11,]   11    2    1
#> [12,]    7    3    1
#> [13,]    8    3    1
#> [14,]    9    3    1
#> [15,]   10    3    1
#> [16,]   11    3    1
#> [17,]   12    3    1
#> [18,]   13    3    1
#> [19,]   14    3    1
#> [20,]   15    4    1
#> [21,]   16    4    1
```

```
#> [22,]   17    4    1
#> [23,]   18    4    1
#> [24,]   19    4    1
#> [25,]   20    4    1
#> [26,]   21    4    1
#> [27,]   22    4    1
#> [28,]   23    4    1
#> [29,]   15    5    1
#> [30,]   16    5    1
#> [31,]   17    5    1
#> [32,]   18    5    1
#> [33,]   24    6    1
#> [34,]   25    6    1
#> [35,]   26    6    1
plot(xyw[, 1:2])
```



6 Como recuperar as informações dos relacionamentos

Vimos, anteriormente, as propriedades das atividades disponíveis. Agora iremos estudar as propriedadas dos relacionamentos. Na sequência, veremos como recuperar os predecessores e os sucessores de uma atividade, ainda saber se um relacionamento é redundante ou não.

6.1 O cronograma possui algum relacionamento? Quantos?

Existem vários métodos sobre os relacionamentos que você pode utilizar para recuperar informações sobre eles. Após incluir atividades e relacionamentos num cronogram, você pode querer saber quantos relacionamentos o cronograma possui ou então, quais desses relacionamentos pertencem ao caminho crítico. Vamos às explicações dessas propriedades.

- **Has Any Relation:** Valor lógico que indica se o cronograma possui algum relacionamento ou não.
  - `TRUE`: indica que o cronograma tem pelo menos um relacionamento;
  - `FALSE`: indica que o cronograma não possui nenhum relacionamento.
  - Utilização: `-sch_has_any_relation(sch)`
- **Number of Relations:** Informa o número de relacionamentos que o cronograma possui.
- Utilização:
  - `sch_nr_relations(sch)`

Exemplos:

```
sch <- cronograma_vazio()

sch_has_any_relation(sch)  # FALSE
#> [1] FALSE
sch_nr_relations(sch)      # 0
#> [1] 0

sch <- cronograma_exemplo()

sch_has_any_relation(sch)  # TRUE
#> [1] TRUE
sch_nr_relations(sch)      # 7
#> [1] 7
```

6.2 Recuperando os relacionamentos

A relação de precedência lógicas entre as atividades está guardada num tibble. Ele é formado pelo identificador da atividade precessora e da sucessora, o tipo e o lag. Segue a descrição de cada uma das colunas do tibble.

- **Relations:** Retorna um tibble com todas os relacionamentos do cronograma, obedecendo a ordem topológica entre eles. A ordem topológica obedece a relação de precedência do diagrama de rede delas. Esta é a outra principal informação calculada pelo CPM.
- Utilização:
  - `sch_relations(sch)`
- O tibble possui a seguinte estrutura:
  - **from:** Identificador da atividade predecessora da relação. Deve existir uma atividade com esse identificador.
  - **to:** identificador da atividade sucessora da relação. Deve existir uma atividade com esse identificador.

  - **type:** Tipo do relacionamento entre as atividades. Seu valor pode ser: FS, FF, SS, SF, significando:
    * `FS`: Término-Início: A atividade sucessora só pode começar depois que a predecessora finalizar.
    * `FF`: Término-Término: A atividade sucessora deve finalizar junto com a predecessora.
    * `SS`: Início-Início: A atividade sucessora deve iniciar junto com a predecessora.
    * `SF`: Início-Término: A atividade sucessora deve finalizar quando a predecessora iniciar.
  - **lag:** Intervalo de tempo que a atividade sucessora deve esperar, ou adiantar, para ser executada. Por exemplo, num relacionamento FS, com `lag = 4`, após a predecessora finalizar, a sucessora deve esperar 4 períodos. Em outras palavras, a atividade sucessora avança 4 períodos após o término da predecessora. Se o valor for negativo, a atividade sucessora antecipa seu início. No exemplo acima, se `lag = -4`, a atividade sucessora adiantaria em 4 períodos o seu início. Por padrão, `lag = 0`.
  - **critical:** Indica se o relacionamento é crítico ou não. Um relacionamento só é crítico se ele for formado por duas atividades críticas.
    * `TRUE`: indica que o relacionamento é crítico;
    * `FALSE`: indica que o relacionamento não é crítico.
  - **ord:** É a ordem em que os relacionamentos foram incluídos no cronograma.
  - **i_from:** Índice da atividade predecessora lá no tibble de atividades.
  - **i_to:** Índice da atividade sucessora lá no tibble de atividades.

```
sch <- cronograma_exemplo()

sch_relations(sch) %>%
  str()
#> tibble [7 x 8] (S3: tbl_df/tbl/data.frame)
#>  $ from    : int [1:7] 1 1 2 3 3 4 5
#>  $ to      : int [1:7] 2 3 4 4 5 6 6
#>  $ type    : chr [1:7] "FS" "FS" "FS" "FS" ...
#>  $ lag     : int [1:7] 0 0 0 0 0 0 0
#>  $ critical: logi [1:7] FALSE TRUE FALSE TRUE FALSE TRUE ...
#>  $ ord     : int [1:7] 1 2 3 4 5 6 7
#>  $ i_from  : int [1:7] 1 1 2 3 3 4 5
#>  $ i_to    : int [1:7] 2 3 4 4 5 6 6
```

6.3 Atividades predecessoras e sucessoras

Um cronograma é formado por atividades e pelo relacionamento entre elas. Isso significa que cada atividade pode ter zero, uma ou mais predecessoras, o mesmo ocorrendo para suas sucessoras. Além disso, uma relação pode ser redundante. Os próximos comandos mostram como recuperar essas informações do cronograma.

- **Successors:** Lista as sucessoras diretas de uma atividade.
- Utilização:
  - `sch_successors(sch, id)`
  - **id**: Identificador da atividade a ser inspecionada.
- **All Successors:** Lista todas as sucessoras diretas e indiretas de uma atividade.
- Utilização:
  - `sch_all_successors(sch, id, ign_to = NULL)`
  - **id**: Identificador da atividade a ser inspecionada.

  - **ign_to**: Relação a ser ignorada na listagem: `id -> ign_to`. Atividades que pertencem a essa relação serão ignoradas. Ele é opcional.

- **Predecessors:** Lista as predecessoras diretas de uma atividade.
- Utilização:
  - `sch_predecessors(sch, id, ign_from = NULL)`
  - **id**: Identificador da atividade a ser inspecionada.

- **All Predecessors:** Lista todas as predecessoras diretas e indiretas de uma atividade.
- Utilização:
  - `sch_all_predecessors(sch, id, ign_from = NULL)`
  - **id**: Identificador da atividade a ser inspecionada.
  - **ign_from**: Relação a ser ignorada na listagem: `ign_from -> id`. Atividades que pertencem a essa relação serão ignoradas. Ele é opcional.

- **Is Redundant:** Verifica se a relação entre duas atividades é redundante. Uma relação `A->C` é redudante se existirem as seguintes relações: `A->B, B->C, A->C`. Ela retorna um valor lógico.
  - `TRUE`: se o relacionamento é redundante;
  - `FALSE`: se o relacionamento não é redundante.

- Utilização:
  - `sch_is_redundant(sch, id_from, id_to)`
  - **id_from**: identificador da atividade predecessora.
  - **id_to**: identificador da atividade sucessora.

- **Evaluate Redundancy:** Avalia a redundância de todas as atividades e cria uma nova coluna no tibble de relacionamentos. Essa operação não é feita automaticamente, você deve chamá-la quando achar necessário.
- Utilização:
  - `sch_evaluate_redundancy(sch)`

Exemplos: Para esses exemplos é bom ter o digrama de rede na cabeça, ou melhor, visível. Então, vou mostrá-lo aqui novamente.

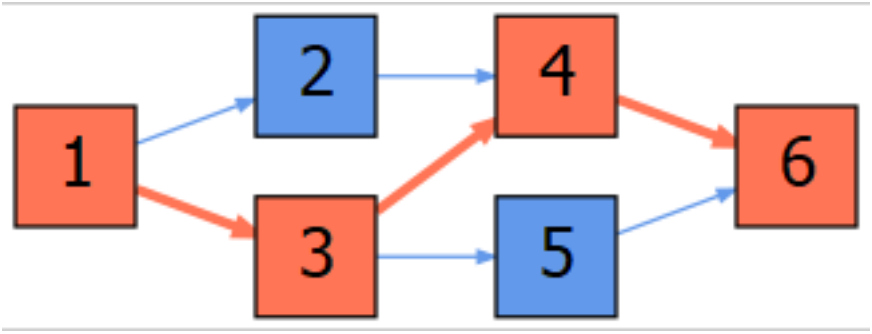


Figure 2: Diagrama de rede

Sucessoras das atividades

```
sch <- cronograma_exemplo()

sch_successors(sch, 2) # 4
#> [1] 4
sch_successors(sch, 3) # 4, 5
#> [1] 4 5
sch_successors(sch, 4) # 6
#> [1] 6

sch_all_successors(sch, 2) # 4, 6
#> [1] 4 6
sch_all_successors(sch, 3) # 4, 5, 6
#> [1] 4 5 6
sch_all_successors(sch, 4) # 6
#> [1] 6
```

Predecessoras das atividades

```
sch <- cronograma_exemplo()

sch_predecessors(sch, 1) # Nenhum
#> integer(0)
sch_predecessors(sch, 3) # 1
#> [1] 1
sch_predecessors(sch, 4) # 2, 3
#> [1] 2 3

sch_all_predecessors(sch, 1) # Nenhum
#> numeric(0)
```

```
sch_all_predecessors(sch, 3) # 1
#> [1] 1
sch_all_predecessors(sch, 4) # 1, 2, 3
#> [1] 3 2 1
```

Avaliação da redundância

```
sch <- cronograma_exemplo()

sch_duration(sch) # 26
#> [1] 26
sch_is_redundant(sch, 2, 4)  #FALSE
#> [1] FALSE

sch %<>%
  sch_add_relation(2L, 6L) %>%
  sch_plan()

sch_duration(sch) # 26, ainda, pois a ligação é redundante.
#> [1] 26
sch_is_redundant(sch, 2, 6) #TRUE
#> [1] TRUE

sch %<>%
  sch_evaluate_redundancy()

# A última coluna da tabela indica que existe apenas um relacionamento
# redudante, justamente o 2->6, que foi o que nós incluímos.
sch_relations(sch)
#> # A tibble: 8 x 9
#>    from    to type    lag critical   ord i_from  i_to redundant
#>   <int> <int> <chr> <int> <lgl>    <int>  <int> <int> <lgl>
#> 1     1     2 FS        0 FALSE        1      1     2 FALSE
#> 2     1     3 FS        0 TRUE         2      1     3 FALSE
#> 3     2     4 FS        0 FALSE        3      2     4 FALSE
#> 4     2     6 FS        0 FALSE        8      2     6 TRUE
#> 5     3     4 FS        0 TRUE         4      3     4 FALSE
#> 6     3     5 FS        0 FALSE        5      3     5 FALSE
#> 7     4     6 FS        0 TRUE         6      4     6 FALSE
#> 8     5     6 FS        0 FALSE        7      5     6 FALSE
```

7 Como recupearar Indicadores topológicos

Os indicadores topológicos mostram informações sobre a estrutura da rede. Eles podem ser de quatro tipos:

- **SP Serial or Parallel:** Mostra o quão próximo é uma rede de ser sequencial ou paralela. À medida que a rede se torna serial, o SP aumenta, até chegar em um, quando ela é totalmente sequencial. À medida que a rede se torna paralela, o SP diminui, até chegar zero, quando ela é totalmente paralela.
- Utilização:
  - `sch_topoi_sp(sch)`

- **AD Activity Distribution:** Mede a distribuição das atividades em seus níveis. Se o AD é aproximadamente igual a zero, a quantidade de atividades de cada nível não é distribuída uniformemente. Se ele for aproximadamente igual a um, cada nível possui uma distribuição mais uniforme, como se todas as atividades possuíssem a mesma quantidade de sucessoras.
- Utilização:
  - `sch_topoi_ad(sch)`
- **LA Length of Arcs:** Mede a presença de arcos longos, baseados na diferença entre o nível progressivel da atividade predecessora e a sua sucessora. Se o LA é aproximadamente igual a zero, o nível progressivo entre as atividades é o mais longe possível. De outro modo, se o LA é igual a um, a distância entre as relações é um.
- Utilização:
  - `sch_topoi_la(sch)`
- **TF Topological Float Indicator:** Mede a folga topológica entre cada atividade. Se TF for igual a zero, não existe folga entre as atividades. Se o TF igual a um, existem folgas entre as atividades e elas podem ser deslocadas sem afetar suas atividades sucessoras.
- Utilização:
  - `sch_topoi_tf(sch)`

```
sch <- cronograma_exemplo()

sch_topoi_sp(sch) # 0.6
#> [1] 0.6
sch_topoi_ad(sch) # 0.6666667
#> [1] 0.6666667
sch_topoi_la(sch) # 0.6666667
#> [1] 0.6666667
sch_topoi_tf(sch) # 0
#> [1] 0
```

Para verificar a mudança nos indicadores topológicos, vamos alterar o cronograma incluindo algumas atividades e relacionamentos.

```
sch <- cronograma_exemplo() %>%
  sch_add_activity(7L, "G", 3L) %>%
  sch_add_activity(8L, "G", 4L) %>%
  sch_add_relation(1L, 7L) %>%
  sch_plan()

sch_duration(sch)
#> [1] 26

sch_topoi_sp(sch) # 0.4285714
#> [1] 0.4285714
sch_topoi_ad(sch) # 0.3333333
#> [1] 0.3333333
sch_topoi_la(sch) # 0.25
#> [1] 0.25
sch_topoi_tf(sch) # 0.4166667
#> [1] 0.4166667

arquivo <- sch %>%
  gerar_arquivo_do_diagrama()
```

O novo diagrama ficou no seguinte formato:

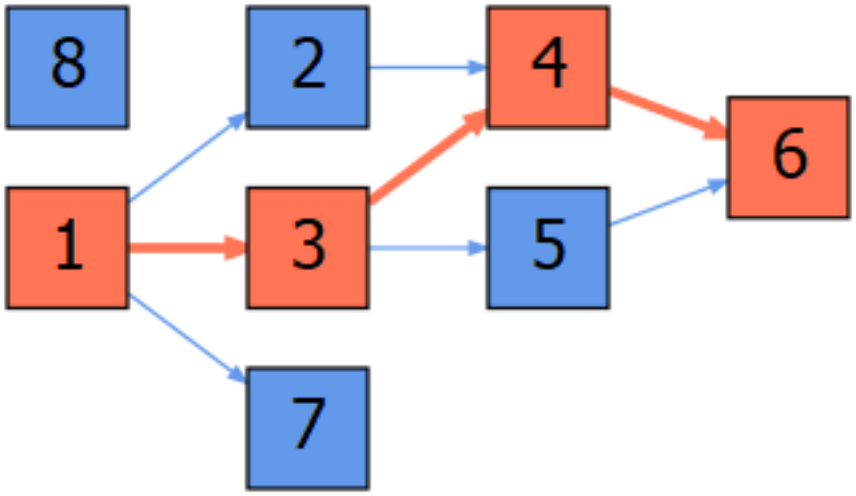


Figure 3: Diagrama de rede

Como os indicadores topológicos refletem o formato da estrutura do cronograma [20], as alterações feitas no diagrama de rede alterarm os seus valores. Mais informações sobre indicadores topológicos podem ser encontradas em [20] e [36].

8 Referências

Segue aqui algumas referências utilizadas neste documento. Para referências completas, você pode acessar o site Análise de Cronogramas.

Bache, S. & Wickham, H. (2022). **magrittr:** A Forward-Pipe Operator for R. Magrittr Tidy Verse, Magrittr GitHub.

Csardi, G. & Nepusz, T. (2005). The Igraph Software Package for Complex Network Research. *InterJournal*. Complex Systems. 1695. Article / igraph

Project Management Institute (2017). **A Guide to the Project Management Body of Knowledge (PMBOK Guide)**. Sixth Edition. PMBOK

Project Management Institute (2017). **PMI Lexicon of Project Management Terms:** Version 3.2. PMI Lexicon

Rosa, Rubens Jose; Santos, Marcos dos; Marques, Thiago (2022). **criticalpath**: An Implementation of the Critical Path Method. R package version 0.2.1. https://rubensjoserosa.com/criticalpath, https://github.com/rubens2005/criticalpath

Vanhoucke, M. (2009) **Measuring Time**: Improving Project Performance Using Earned Value Management. Springer-Verlag US. doi://10.1007/978-1-4419-1014-1.

Vanhoucke, M. (2013) **Project Management with Dynamic Scheduling**: Baseline Scheduling, Risk Analysis and Project Control. Springer-Verlag Berlin Heidelberg. doi://10.1007/978-3-642-40438-2

Vanhoucke, M. (2014) **Integrated Project Management and Control**: First Comes the Theory, then the Practice. Springer International Publishing Switzerland. doi://10.1007/978-3-319-04331-9

9 Autores do pacote `criticalpath`

9.1 Rubens José Rosa

E-mail: rubens@rubensjoserosa.com

Site: https://rubensjoserosa.com/

Figure 4: Rubens José rosa

Mestre em Gestão da Informação (UFPR), Especialista em Gerenciamento de Projetos (FGV), Especialista em Linguagem Java (Universidade Positivo), Especialista em Desenvolvimento Web (PUCPR), Bacharel em Processamento de Dados (UEPG). Experiência profissional de 20 anos na Copel, como Analista de Sistemas na Superintendência de Tecnologia da Informação e, desde 2012, como Analista de Gestão Estratégica, atuando no Escritório de Projetos Corporativos – EPC e no Sistema de Gestão Estratégica.

9.2 Marcos dos Santos

E-mail: marcosdossantos@ime.eb.br

Site: https://casadapesquisaoperacional.com/

Figure 5: Marcos dos Santos

Capitão de Fragata da Marinha do Brasil, com 28 de anos de carreira. Há 11 anos trabalha como pesquisador do Centro de Análises de Sistemas Navais (CASNAV) assessorando a Alta Administração Naval em processos decisórios de elevada complexidade. Possui curso de Altos Estudos Militares pela Escola de Guerra Naval (EGN). É Pesquisador de pós-doutorado em Ciências e Tecnologias Espaciais pelo Instituto Tecnológico de Aeronáutica (ITA). Doutor em Sistemas, Apoio à Decisão e Logística pela Universidade Federal Fluminense (UFF). Mestre em Pesquisa Operacional pela COPPE UFRJ. É Professor de Matemática Avançada do Instituto Militar de Engenharia (IME). Também faz parte do corpo docente do curso de Data Science e Analytics da Universidade de São Paulo (USP). Faz parte da Diretoria da Sociedade Brasileira de Pesquisa Operacional (SOBRAPO). Possui mais de 500 pesquisas publicadas, dentre artigos científicos, capítulos de livros, patentes, registros de softwares, códigos no CRAN e outros trabalhos técnicos.

9.3 Thiago Marques

E-mail: profestathimarques@gmail.com

Site: http://estatidados.com.br/

Figure 6: Thiago Marques

Estatístico pela federal do IBGE, a ENCE (Escola Nacional de Ciências Estatísticas), entusiasta na disseminação da Estatística e possui larga experiência na atuação em Estatística no mercado, tendo passado por grandes centros universitários, multinacionais de consultoria, pelo IBRE/FGV (Instituto Brasileiro de Economia) e pelo IBGE (Instituto Brasileiro de Geografia e Estatística), Foi professor da pós de ciências de dados do SENAC RJ e da UNISUAM e é o criador do maior canal de Estatística, Ciência de dados da América Latina! O chamado EstaTiDados :). Atualmente é Consultor em Análise Estatística no IBGE e professor na sua Comunidade de Estatística, na Casa da Pesquisa Operacional e professor da MBA em Ciência de dados da UNIFOR, MBA em Ciência de dados da Farias Brito, da pós-graduação em BI com ênfase em Ciências de dados do ICEV e da MBA em Data Science e Analytics da USP, realiza palestras e treinamentos para capacitação em Estatística, Apoio a tomada de decisão, Pesquisa Operacional, R e Ciência de dados.