

Timeliner: uma heurística para compactar e organizar eventos em linhas de tempo.

Bruno Gonçalves¹ e João Batista Oliveira¹.

¹Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

bruno.selau@edu.pucrs.br, oliveira@inf.pucrs.br

Resumo.

1. Introdução

Este projeto trata da criação de um organizador de eventos em uma linha do tempo a partir de uma entrada de dados. Existem vários programas que criam gráficos para esta finalidade *Time.Graphics*[tim], mas a utilização desses *softwares* ainda demanda alguma quantidade de tempo e dedicação dos usuários. A motivação desta aplicação visa facilitar a organização de eventos para profissionais como professores, gerentes de projeto, *designers*, estudantes, administradores e pesquisadores, entre outros, pois ao eliminar a necessidade de dedicar tempo fica mais rápido e atraente criar a sua organização sem precisar escrever em um quadro ou arrastar eventos na tela do computador.

[Neufeld and Tartar 1974] mostram a redução do problema do número cromático para o problema de determinar um cronograma de uma faculdade, que acaba por ser mais complexo que o deste projeto por possuir mais restrições. Mostraremos que o problema menor, da colocação de itens em linhas do tempo também pode ser mapeado (reduzido) para o problema do número cromático de um grafo se pensarmos em cada evento como um vértice e cada linha como uma cor do grafo. O algoritmo aqui proposto busca uma aproximação do número cromático. Encontrar uma coloração mínima é um dos 21 problemas NP-completos de [KARP 1967], sendo classificado como NP-Difícil, que neste caso significa ser inviável utilizar *backtracking* para testagem de todos os resultados possíveis, porque conforme as quantidades de vértices e arestas crescem aumenta demasiado o consumo de tempo e memória. Sabendo destes desafios o algoritmo implementado neste projeto é de caráter guloso, para que o algoritmo não use muito tempo para organizar o gráfico do usuário. O *Timeliner* não é um algoritmo que resolve todos os problemas relacionados com coloração de grafos, mas será comparado com os algoritmos *Largest-First* e *DSatur* que são genéricos para problemas de número cromático.

O algoritmo implementado em python começa ao receber eventos a partir de uma entrada de dados em um arquivo *.txt*: cada evento possui uma nome, data inicial e data final. Então são utilizadas ordenação e operações sobre conjuntos para colocar os eventos em uma distribuição no gráfico, gerando uma imagem na saída. Para validação e comparação será utilizada a ferramenta *Sagemath*[Sag] que já possui seu algoritmo para calcular o número cromático, e permitiu a implementação de uma versão do *Largest-First* ao decorrer deste projeto, com fins de comparação ao *Timeliner*.

2. Trabalhos relacionados

Os trabalhos apresentados nesta seção serviram de base para a projeto que resolverá o problema através de coloração ordenação e operações sobre conjuntos. O problema a ser

resolvido é equivalente a coloração de grafos que é classificado como NP-Difícil, então até o momento não existe um algoritmo que resolva estes problemas em tempo polinomial e ao mesmo tempo garanta uma solução ótima global.

Existem várias soluções para criar linhas do tempo, dentre elas o *Time.Graphics*[tim], que é uma aplicação *web* já pronta e amplamente usada, sendo uma ferramenta próxima daquilo que é proposto neste artigo devido a sua aparência e ao fato de não permitir sobreposição de intersecções, mas toda organização deve ser feita manualmente.

Apesar do *Time.Graphics*[tim] ter pontos relevantes, não há um bom objeto de comparação para este projeto, pois para isso é necessário um foco em organização e compactação como o deste projeto. Com isso o trabalho de pesquisa foi essencial para chegar em um bom resultado, buscando artigos que abordam resoluções de problemas combinatórios, mas restringindo nosso pensamento a abordagens gulosas. A relação entre o problema e a coloração de grafos apenas foi descoberta durante a pesquisa, e nesta mesma etapa a ferramenta *Sagemath*[Sag] apareceu como uma ótima opção para a etapa de experimentação.

2.1. Reducibility among Combinatorial Problems

Este artigo aborda a teoria da complexidade computacional tratando de um conjunto de problemas computacionais que envolvem combinatória e grafos, e é conhecido como os "21 problemas de Karp". O autor[KARP 1967] usou o teorema de que o problema da satisfatibilidade é NP-completo, mostrando uma redução por mapeamento em tempo polinomial do problema de satisfatibilidade para cada problema computacional de seu artigo.

O artigo evidenciou que muitos problemas da computação podem ser intratáveis para uma resposta exata, e um dos 21 problemas reduzidos por Karp é o problema do número cromático ou também conhecido como coloração de grafos, que é o problema abordado neste artigo. O problema de coloração de grafos é um dos problemas que não é possível encontrar a resposta ótima para alguns casos, sendo classificado alguns anos depois[?] como NP-Difícil.

2.2. Graph coloring conditions for the existence of solutions to the timetable problem

Este artigo[Neufeld and Tartar 1974] que foi publicado na ACM(*Association for Computing Machinery*), falando sobre o problema do cronograma do professor. No artigo é descrito o problema do cronograma do professor e como este pode ser resolvido a partir da coloração de grafos. O artigo mostra que o cronograma de um professor é uma linha do tempo assim como as que tratamos na implementação descrita neste artigo, mas no caso deste problema os eventos possuem diversos casos de restrição, sendo então um caso mais complexo.

2.3. An upper bound for the chromatic number of a graph and its application to timetabling problems

Neste artigo os autores[Welsh and Powell 1972] trazem uma observação mostrando que os vértices com maior grau são mais difíceis de se colorir, se comparado aos vértices de menor grau. Esta dificuldade ocorre pois possuir maior grau significa possuir mais

ligações, e com mais ligações torna-se menor o conjunto de cores disponíveis. Ao colorir os vértices de maior grau teremos mais flexibilidade, pois evitamos as intersecções.

Este é um artigo matemático que propôs um novo limite superior para o número cromático de um gráfico, e além disso traz um novo algoritmo guloso que trabalha dentro deste limite. O algoritmo é conhecido como *largest-first* ou LF, pois os vértices de maior grau são coloridos primeiro.

O primeiro passo do algoritmo é ordenar os vértice de forma decrescente, e então associe a cor 1 ao vértice de maior grau e ao próximo vértice da lista não adjacente a ele, e sucessivamente para cada nó não adjacente aos vértices da cor 1. Então faça o mesmo processo para a cor 2, ignorando os vértices ligados e já coloridos.

Algorithm 1 Largest-First

```

1: vértices.ordenaPorGrau()                                ▷ ordenação pela quantidade de arestas
2: listaCores  $\leftarrow$  [[vértices[0]]]                                ▷ Lista das Cores
3: vértices.remove(0)                                ▷ Remove o primeiro elemento
4: para cada i em vértices
5:   para cada j em listaCores
6:     verificaLigacao  $\leftarrow$  0
7:     para cada k em j
8:       se i é ligado com k então
9:         verificaLigacao  $\leftarrow$  1
10:        break
11:      fim se
12:    fim para cada
13:    se verificaLigacao < 1 então
14:      j.adiciona(i)                                ▷ adiciona o nodo i na cor j
15:      break
16:    fim se                                ▷ Remove o primeiro elemento
17:  fim para cada
18:  se verificaLigacao > 0 então
19:    listaCores.adiciona([i])                                ▷ Cria nova cor
20:  fim se
21: fim para cada
22: devolve listaCores, tamanho(listaCores)

```

Este algoritmo é guloso assim como o apresentado neste artigo, ou seja, é uma solução rápida para resolver o problema, e encontra uma boa aproximação de número cromático para qualquer problema de grafos. Porém a relacionar com o *Timeliner*, na seção 4, podemos perceber que o *Largest-First* necessita verificar as ligações em dois momentos. O primeiro momento é durante a ordenação, pois esta é feita pela quantidade de graus, sendo necessário verificar todas as ligações de nodos para que seja possível começar a ordenação. O algoritmo aqui proposto não faz esta primeira verificação, pois sua ordenação é feita pela duração de um evento. A segunda verificação é igual para os dois algoritmos e acontece no momento da inserção das cores ou linhas.

2.4. New methods to color the vertices of a graph

O algoritmo *DSatur*[Brélaz 1979] é um algoritmo com carácter dinâmico, diferente do *Largest-First* que é guloso. Este algoritmo é exato para grafos bipartidos e importante para buscar cliques maximais em grafos. Goldberg e Goldberg (2012) demonstram que o algoritmo *DSATUR* tem tempo $O(n^2)$.

O primeiro passo é definir o primeiro vértice colorido que será o de maior grau, os outros vértices serão escolhidos pelo maior grau de saturação. Ou seja, o vértice com o maior número de diferentes cores adjacentes será colorido, e este será colorido com a cor de menor índice não utilizada por seus vizinhos. Se houver empate o critério de desempate é o grau do vértice, se mesmo assim houver empate o vértice será escolhido forma aleatória.

Em razão do tempo não foi possível implementar o *DSATUR*, o que já é sabido deste em relação ao *Timeliner* é que o seu tempo é maior pois o seu carácter dinâmico implica que todos vértices devem ser visitados. À custa deste mesmo fato é feita na seção 6 a seguinte pergunta: Existem casos que o numero cromático encontrado pelo *DSATUR* seja menor que encontrado pelo algoritmo do artigo?

2.5. Chromatic scheduling and the chromatic number problem.

Este artigo[Brown 1972] trouxe a primeira resolução para o problema de coloração dos vértices. Este algoritmo é ótimo para resolver o problema, porém estamos falando de um problema NP-Difícil e conforme o dados do problema crescem torna-se impossível que o algoritmo termine de rodar.

Este também é um algoritmo que a sua implementação ficará como sugestão de trabalho futuro, pois além de compreender o o funcionamento a implementação permitira saber até onde um solução ótima consegue chegar. A função de número cromático disponibilizada pelo *Sagemath*[Sag] não aguenta casos muito grandes, logo existe a possibilidade de esta função ser uma versão do algoritmo de Brown. Além disso, ao implementar o algoritmo de Brown será possível saber a veracidade dos resultados do *Timeliner*.

3. Modelo

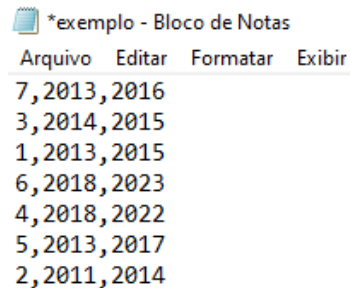
O algoritmo proposto organiza eventos em um gráfico e foi construído partindo da seguinte hipótese: Os eventos que são formados por um grande intervalo possuem mais chances de possuir conflito de datas, pois ocuparão uma parte maior de uma linha no gráfico. Então os eventos com maior duração devem ser alocados primeiro e os menores devem se adaptar buscando os lugares restantes.

A implementação do algoritmo foi construída em python e pode ser dividida em 3 etapas que são: a entrada e o tratamento de dados, organização dos eventos e por ultimo o *plot* do gráfico. Além do passos do algoritmo mostraremos a redução do problema do número cromático para o desafio a ser resolvido.

3.1. Entrada e tratamento de dados

A entrada de dados é feita a partir de um arquivo de texto, em razão da simplicidade de escrever e extrair dados deste tipo de arquivo. Para isso, cada linha do arquivo de texto corresponde à um evento diferente, que possui os seguinte atributos: nome, data de inicio

e data final. Os atributos de um eventos são separado por virgula, como demonstra a Figura 1, a Figura 2 demonstra que os nomes dos eventos também podem ser textos e não somente números como na Figura 1.

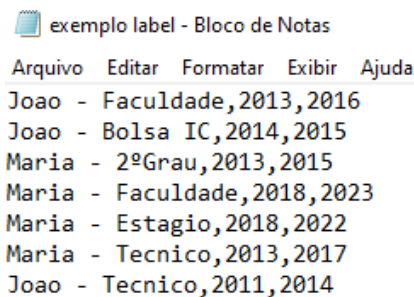


```
*exemplo - Bloco de Notas
Arquivo  Editar  Formatar  Exibir
7,2013,2016
3,2014,2015
1,2013,2015
6,2018,2023
4,2018,2022
5,2013,2017
2,2011,2014
```

Figura 1. Entrada de dados

O primeiro passo do algoritmo é criar uma lista e inserir os eventos enquanto ocorre a leitura do documento de texto, cada linha deste documento será um evento que possui seus atributos divididos por virgula. A fim de aproveitar a iteração sobre a lista, um novo atributo que não esta presente no arquivo de entrada é criado em execução. Este novo atributo é o tamanho do intervalo de cada evento, ou seja, $\text{intervalo} = \text{data final} - \text{data inicial}$.

Durante a criação da lista de eventos são guardadas duas datas. Chamaremos a primeira data de *first*, pois ela guardara a data inicial do evento que começou primeiro, e a segunda data será *last*, pois guardara a data final do evento que terminou por ultimo. Estas duas variáveis servirão para definir os limites do *plot* do gráfico.



```
exemplo label - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
Joao - Faculdade,2013,2016
Joao - Bolsa IC,2014,2015
Maria - 2ºGrau,2013,2015
Maria - Faculdade,2018,2023
Maria - Estagio,2018,2022
Maria - Tecnico,2013,2017
Joao - Tecnico,2011,2014
```

Figura 2. Entrada de dados com label

3.2. Coloração de grafos

Coloração de grafos ou problema do número cromático é um dos 21 problemas de Karp que tem como objetivo rotular os vértices de um grafo. Para compreender o problema é necessário saber que um grafo é formado por vértices e arestas. Os vértices são representados por círculos como na Figura 3, e as arestas são as linhas que ligam os vértices.

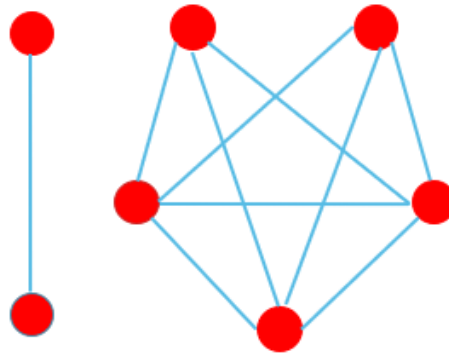


Figura 3. Grafo sem coloração

Um grafo colorido corretamente, conforme o problema do número cromático, deve satisfazer a seguinte regra: dois vértices adjacentes não podem compartilhar a mesma cor, ou seja, dois vértices ligados por uma aresta devem possuir cores (ou rótulos) diferentes. Com isso podemos definir o número cromático de um grafo como sendo a quantidade mínima de cores que colorem os vértices de um grafo satisfazendo esta condição. Logo, encontrar o número cromático significa encontrar a menor quantidade possível de cores para colorir um grafo de forma que vértices vizinhos não tenham a mesma cor.

A Figura 3 representa exatamente o mesmo grafo que a Figura 4. A diferença entre as duas imagens deve-se a coloração de grafos, pois a Figura 4 possui um grafo colorido de acordo com a regra do número cromático. Para transformar (reduzir) o problema da coloração de grafos para o problema das linhas do tempo e dos conflitos de data é necessário formular as seguintes associações:

- Um vértice de um grafo corresponde a um evento no gráfico;
- Uma aresta de um grafo corresponde a um conflito de data;
- Uma cor de um vértice do grafo corresponde a uma altura na linha de tempo.

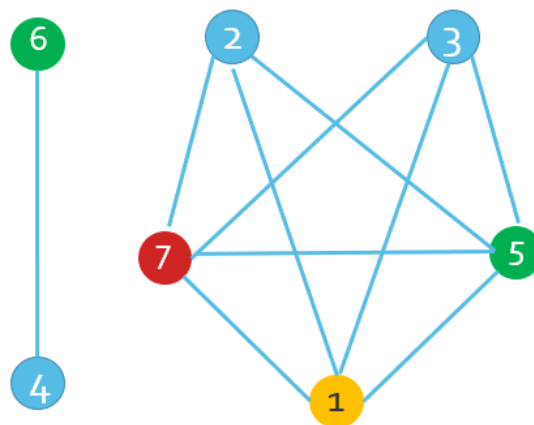


Figura 4. Coloração de grafos

A Figura 5 é a imagem gerada pelo *Timeliner*, e também é a representação em uma linha do tempo do mesmo problema e domínio do grafo das Figuras 3 e 4. Com o objetivo de facilitar a associação das representações, os vértices e eventos estão numerados nas Figuras 4 e 5. Para saber se o grafo (Figura 4) é uma fiel representação da linha do tempo (Figura 5) é preciso fazer as seguintes perguntas:

- O número de vértices do grafo é igual ao número de eventos do gráfico?
 - Sim, o grafo possui 7 vértices e o gráfico 7 eventos.
- Os vértices adjacentes ao vértice n , correspondem aos eventos que possuem conflito de data com o evento n ?
 - Sim;
 - Os vértices adjacentes ao vértice 1 são 2,3,5 e 7, assim como os eventos que possuem conflito de data com o evento 1 são 2,3,5 e 7;
 - Os vértices adjacentes ao vértice 2 são 1,5 e 7, assim como os eventos que possuem conflito de data com o evento 2 são 1,5 e 7;
 - Os vértices adjacentes ao vértice 3 são 1,5 e 7, assim como os eventos que possuem conflito de data com o evento 3 são 1,5 e 7;
 - Apenas o vértice 6 é adjacente ao vértice 4, assim como o evento 6 é o único que possui conflito de data com o evento 4;
 - Os vértices adjacentes ao vértice 5 são 1,2,3 e 7, assim como os eventos que possuem conflito de data com o evento 5 são 1,2,3 e 7;
 - Apenas o vértice 4 é adjacente ao vértice 6, assim como o evento 4 é o único que possui conflito de data com o evento 6;
 - Os vértices adjacentes ao vértice 7 são 1,2,3 e 5, assim como os eventos que possuem conflito de data com o evento 7 são 1,2,3 e 5;

- O numero cromático do gráfico é igual a quantidade de linhas do gráfico?
 - Sim, o numero cromático do gráfico é 4, que é igual quantidade de linhas que os eventos estão distribuídos no gráfico.

Com esse exemplo é possível perceber que ao mesmo tempo que os conflitos de linha do tempo são resolvidos, o problema de coloração de grafos esta sendo resolvido.



Figura 5. Linha do tempo

3.3. Organização dos eventos

O *Timeliner* começa na etapa da entrada de dados, mas o diferencial deste algoritmo esta na organização de eventos pois não é necessário calcular o grau de cada evento no inicio. O *Largest-First* calcula o grau de cada um dos vértices do grafo para que seja possível ordenar a lista de vértices, e então começar a colorir primeiro os vértices de maior grau. O *Timeliner* também precisa ordenar os eventos, mas o seu critério não é o grau e sim a duração dos eventos, diminuindo a quantidade de operações para ordenação ao se comparar com o *Largest-First*.

Algorithm 2 Timeliner

```
1: eventos.ordenaTamanhoDoIntervalo()    ▷ ordenação pelo tamanho do intervalo
2: listaLinhas  $\leftarrow$  [eventos[0]]        ▷ Lista das Linhas do gráfico
3: eventos.remove(0)                       ▷ Remove o primeiro elemento
4: para cada i em eventos
5:   para cada j em listaLinhas
6:     verificaIntersecção  $\leftarrow$  0
7:     para cada k em j
8:       se i possui conflito de data com k então
9:         verificaLigacao  $\leftarrow$  1
10:        break
11:      fim se
12:    fim para cada
13:    se verificaLigacao < 1 então
14:      j.adiciona(i)                      ▷ adiciona o nodo i na cor j
15:      break
16:    fim se                                ▷ Remove o primeiro elemento
17:  fim para cada
18:  se verificaLigacao > 0 então
19:    listaLinhas.adiciona(i)              ▷ Cria Linha com evento i
20:  fim se
21: fim para cada
22: devolve listaCores, tamanho(listaCores)
```

Após isso é criada uma lista de linhas, a qual a sua primeira linha contém o primeiro evento da lista de eventos, que devido a ordenação é o evento com a maior duração. Então este mesmo evento é removido da lista de eventos.

Definindo os elementos de *listaLinhas* como $[l_1, l_2, l_3, \dots, l_n]$, então se inicia o processo de definir a posição de cada evento. Para cada evento é necessário verificar se este possui conflito de data com algum evento de l_1 . Se existir conflito, repetiremos o processo em $l(i + 1)$, se não existir conflito o evento pode ser adicionado a esta linha. Caso o evento tenha um conflito de data em todas as linhas, deverá ser adicionada na lista de linha uma nova linha contendo este evento. Esse processo deve ser feito para todos os eventos.

3.4. Plot da linha do tempo

A construção do gráfico depende inteiramente da etapa de organização dos eventos, pois a *listaLinhas* construída nesta etapa anterior é o que define a posição de cada evento no gráfico. Cada elemento da *listaLinhas* é uma lista de eventos sem nenhum conflito de datas, isso permite estruturar uma linha do tempo sem sobreposições.

Seguindo a definição feita na seção anterior, *listaLinhas* é igual a $[l_1, l_2, l_3, \dots, l_n]$. Para cada l_i em *listaLinhas* verificaremos o índice de l_i . Se o índice de l_i é ímpar, os eventos desta lista devem ser impressos na primeira linha disponível acima do eixo x, caso o índice de l_i seja par, os eventos desta lista devem ser impressos na primeira linha disponível abaixo do eixo x. Ao observar a Figura 6 podemos observar que todo l_i de índice ímpar está acima do eixo x, e o inverso acontece com cada l_i de índice par.



Figura 6. Linha do tempo com Label

4. Resultados experimentais

Nesta seção serão apresentados os dados obtidos ao aplicar o algoritmo *Timeliner*, além de mostrar como seriam os resultados do algoritmo se começássemos pelos menores eventos. Utilizaremos a ferramenta *Sagemath*[Sag] com o objetivo de validar os resultados, pois além de permitir implementar o *Sagemath*[Sag] de forma rápida, a ferramenta também disponibiliza uma função para obter o número cromático.

4.1. Exemplo do artigo

O primeiro exemplo testado é um grafo de 7 vértices e 10 arestas, que é o grafo correspondente as figuras apresentadas ao decorrer do artigo. Os 4 algoritmos apresentam o mesmo número cromático, e neste caso o *Timeliner* original leva vantagem sobre os demais, apresentando a menor quantidade total de operações, uma a menos que o *Timeliner* que é ordenado de forma crescente.

O tempo do *Timeliner* é maior que os demais, e isso vai se repetir nos próximos exemplos. Contudo veremos mais para frente que este não é um defeito do algoritmo, mas já é possível saber disso ao lembrar que algoritmos estão sendo rodados em ambientes diferentes. Além disso, não sabemos como é a estrutura dos grafos no *Sagemath*[Sag].

Métodos	Cores/Linhas	Calculo Vizinhos	Inserção	Total	Tempo
Timeliner	4	7	18	25	0.0059
Timeliner(crescente)	4	7	19	26	0.008
Largest-First	4	49	18	67	0.003
<i>Sagemath</i> [Sag]	4	-	-	-	0.00057

4.2. 5 eventos e 6 conflitos

Este caso utilizamos um grafo de 5 vértices e 6 arestas, e todos algoritmo retornam 3 como o número cromático do grafo. As duas versões do *Timeliner* levam vantagem no quesito operações, e desvantagem no tempo.

Métodos	Cores/Linhas	Calculo Vizinhos	Inserção	Total	Tempo
Timeliner	3	5	7	12	0.0057
Timeliner(crescente)	3	5	7	12	0.006
Largest-First	3	25	7	32	0.002
<i>Sagemath</i> [Sag]	3	-	-	-	0.041

4.3. 5 eventos e 9 conflitos

Este caso utilizamos um grafo de 5 vértices e 9 arestas, e todos algoritmo retornam 4 como o número cromático do grafo. As duas versões do *Timeliner* levam vantagem no quesito operações, e desvantagem no tempo.

Métodos	Cores/Linhas	Calculo Vizinhos	Inserção	Total	Tempo
Timeliner	4	5	9	14	0.0083
Timeliner(crescente)	4	5	9	14	0.005
Largest-First	4	25	9	34	0.0001
<i>Sagemath</i> [Sag]	4	-	-	-	0.001

4.4. 20 eventos e 40 conflitos

Este caso utilizamos um grafo de 20 vértices e 40 arestas, e todos algoritmo retornam 5 como o número cromático do grafo. As duas versões do *Timeliner* levam vantagem no quesito operações, e desvantagem no tempo.

Métodos	Cores/Linhas	Calculo Vizinhos	Inserção	Total	Tempo
Timeliner	5	20	104	124	0.019
Timeliner(crescente)	5	20	124	109	0.024
Largest-First	5	400	104	504	0.006
<i>Sagemath</i> [Sag]	5	-	-	-	0.009

4.5. 200 eventos e 2500 conflitos

Este é o maior caso testado com esses algoritmos e é o que mais demonstra diferenças. O algoritmo com a menor quantidade de operações é o *Timeliner* original, nos casos anteriores já ganhava do *Largest-First* e do algoritmo do *Sagemath*[Sag], mas nesse caso ele também superior é ao *Timeliner* crescente.

Ao analisarmos a coluna referente as linhas, o *Timeliner* crescente um número maior de cores que o *Timeliner* original e *Largest-First*, ou seja, a escolha de trazer os eventos de maior duração se mostrou uma boa escolha neste caso. A quantidade de cores do algoritmo do *Sagemath*[Sag] não aparece na tabela pois o algoritmo não conseguiu chegar na resposta. Isso nos possibilita pensar que este algoritmo utiliza *backtracking*, e se isto for verdade, em todos os casos anteriores o *Timeliner* encontrou de fato o número cromático e não apenas uma aproximação. Além disto, também podemos pensar em uma possível associação do ganho em tempo dos casos anterior a ferramenta *Sagemath*[Sag], e não aos algoritmos, pois é muito difícil um algoritmo que *backtracking* ganhar de um guloso nas mesmas condições.

Métodos	Cores/Linhas	Calculo Vizinhos	Inserção	Total	Tempo
Timeliner	40	200	2182	2382	0.082
Timeliner(crescente)	45	200	2320	2520	0.091
Largest-First	40	10000	2182	12182	0.007
<i>Sagemath</i> [Sag]	-	-	-	-	-

5. Considerações finais

O algoritmo construído neste projeto encontra uma aproximação para o número mínimo de linhas em um gráfico de linha do tempo, e como demonstrado na seção 3.2, estamos reduzindo o problema do número cromático ao mesmo tempo que resolvemos o problema em questão. O nossa heurística é aplicável somente ao contexto das linhas do tempo, diferente dos algoritmos citados nos trabalhos relacionados. Mas a heurística apresentada neste projeto consegue encontrar o número cromático em muitos casos, além de utilizar menos operações de comparação do que outros algoritmos gulosos como o *Largest-First*. Ainda não é possível afirmar quais casos o *Timeliner* não chega na resposta ótima, mas ao pensar sobre a proposta inicial a heurística atinge seus objetivos, pois obtém resultados aproximados(experimentações foram exatas) de forma rápida, além de gerar da linha do tempo.

6. Trabalhos futuros

O projeto deixou varias perguntas soltas, como ideias para lapidar a heurística e perguntas que precisam ser respondidas. Durante a etapa da experimentação foi possível perceber

que há situações que podem comprometer o desempenho, que podem aumentar a quantidade de operações comparativas, como eventos grandes com poucas intersecções, e este é um ponto importante que deve ser respondido. Ficaram em aberto outras perguntas em relação ao *Sagemath*[Sag], como: a função de coloração deste *software* utiliza *backtracking*? Esta ferramenta realmente possui uma otimização em suas operações com grafos?

Mas ao final da implementação também foram notados pontos que podem ser melhorados no algoritmo, como o fato de deixar a estrutura dos dados mais leve, evitando uso de listas e focando em estruturas como *map*. Além do fato de adicionar parâmetros como mês e dia às datas dos eventos. Apesar do algoritmo procurar a melhor resposta possível, não está descartada a ideia de criar um editor para usuários sobre a linha do tempo já construída.

Referências

- MS Windows NT kernel description. <https://time.graphics/>. Accessed: 2020-07-02.
- MS Windows NT kernel description. <https://www.sagemath.org/>. Accessed: 2020-07-02.
- Brown, J. R. (1972). Chromatic scheduling and the chromatic number problem. *Management Science*, 19:456–463.
- Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, 22.
- KARP, R. M. (1967). Reducibility among combinatorial problems. *The Computer Journal*, 10:85–86.
- Neufeld, G. and Tartar, J. (1974). Graph coloring conditions for the existence of solutions to the timetable problem. *Communications of the ACM*, 17:450–453.
- Welsh, D. J. A. and Powell, M. B. (1972). An upper bound for the chromatic number of a graph and its application to timetabling problems. *Complexity of Computer Computations*, pages 85–103.