

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`

Abstract. *This article describes an undergraduate final project for the PUCRS computer science course. Throughout this research, the problem of timelines was investigated, with the aim of implementing a heuristic that creates a compact distribution in a graph automatically and quickly. Based on this objective, the reduction of the chromatic number of a graph was elaborated for the time-lines problem, in order to compare our heuristics with heuristics that solve this graph problem.*

Resumo. *Este artigo descreve um trabalho de conclusão de curso da graduação de ciência da computação da PUCRS. Ao longo desta pesquisa foi investigado o problema das linhas do tempo, com o objetivo de implementar uma heurística que cria uma distribuição compacta em um gráfico de forma automática e rápida. A partir deste objetivo foi elaborada a redução do número cromático de um grafo para o problema das linhas do tempo, com o intuito de comparar a nossa heurística com heurísticas que resolvem este problema de grafos.*

1. Introdução

Este projeto trata da criação de um organizador de eventos em uma linha do tempo a partir de uma entrada de dados, pois apesar de existirem vários programas que criam gráficos para esta finalidade, 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, pesquisadores, entre outros. Pois ao eliminar a necessidade de dedicação e tempo fica mais rápido e atraente criar a sua *timeline* sem precisar escrever em um quadro ou arrastar eventos na tela do computador.

Neufeld e Tartar [2] mostram a redução do problema do número cromático (ou coloração de grafos) para o problema de determinar um cronograma de uma faculdade, mas este problema acaba por ser mais complexo que o deste projeto por possuir mais restrições. A seguir mostraremos que o problema do número cromático de um grafo pode ser mapeado (reduzido) para o problema das *timelines*. Com isso, mostramos que estamos tratando com um problema NP-Difícil, pois o problema do número cromático é classificado como NP-Difícil por Garey e Johnson [3].

O algoritmo aqui proposto busca uma organização sem sobreposição de eventos com a menor quantidade possível de linhas, mas isso deve ser feito em um tempo viável. Sabendo que o problema é NP-Difícil, torna-se inviável utilizar *backtracking* para encontrar um resultado exato através de força bruta, porque conforme a quantidade de vértices

e arestas cresce, aumenta demasiado o consumo de tempo. Sabendo dos desafios o algoritmo usado neste projeto é de caráter guloso, com objetivo de não utilizar muito tempo para organizar o gráfico do usuário, podendo até obter uma solução que não seja ótima. Os códigos fonte deste artigo podem ser encontrados [aqui](#).

2. Problema

Para compreender o problema devemos enxergar um evento como um dupla composta por data de início e data final, sendo representado como $e = (inicio, final)$. Ao analisar a Figura 1, podemos construir a representação do evento A que começa no ano de 1003 e termina em 1006, com isso temos que $A = (1003, 1006)$. Uma *timeline* pode ser representada como $T = (E, S)$, onde E é um conjunto de eventos e S é um conjunto de sobreposições. A objetivo de um *Timeliner* é receber uma *timeline* e gerar o conjuntos de linhas (L) de forma que cada linha possua eventos que não se sobreponham.

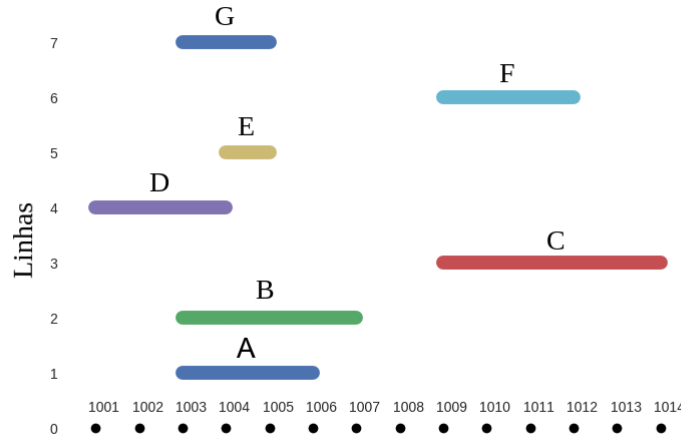


Figura 1. *Timeline* sem organização

Na Figura 1 podemos ver a representação de 7 retas coloridas, as quais chamaremos de eventos, então nosso conjunto é representado como $E = \{A, B, C, D, E, F, G\}$. O segundo conjunto é o das sobreposições, que definiremos como $S = \{(A, B), (A, D), (A, E), (A, G), (B, D), (B, E), (B, G), (C, F), (D, G), (E, G)\}$. As sobreposições são formadas por uma dupla de eventos que podem se sobrepor caso estejam na mesma linha, isso ocorre quando eventos compartilham datas, lembrando que as datas estão representadas no eixo X da Figura 1. Podemos descobrir a quantidade de linhas a partir da cardinalidade de L , que nesta *timeline* esta representada como $L = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}, \{G\}\}$, com isso sabemos que a *timeline* possui 7 linhas, pois sua quantidade de conjuntos é dada como $|L| = 7$. Isso ocorre pois cada evento está em uma altura diferente to gráfico, logo a quantidade de linhas é igual a quantidade de eventos.

Para que seja possível propor um algoritmo que encontre uma boa distribuição e organização de eventos, deve-se evitar a sobreposição de eventos e ao mesmo tempo buscar uma distribuição com a menor quantidade de linhas possível. A Figura 1 não apresenta

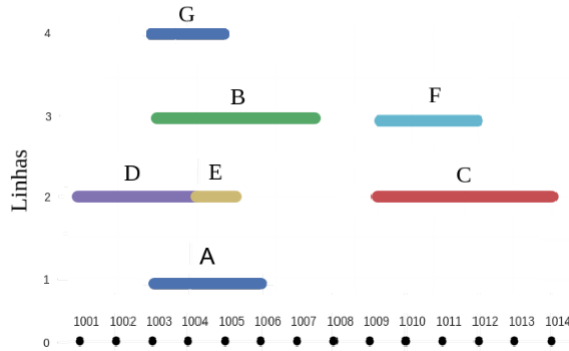


Figura 2. Timeline com organização

uma distribuição otimizada, pois poderia estar organizada de forma mais compacta, como a *timeline* da Figura 2. Esta nova *timeline* possui eventos (E) e intersecções (I) iguais ao exemplo anterior, mas podemos concluir que esta *timeline* está mais compacta ao analisar o conjunto de linhas deste gráfico, que é dado por $L = \{\{A\}, \{D, E, C\}, \{B, F\}, \{G\}\}$. Com isso, é possível compreender que as linhas não possuem apenas 1 evento, e agora possuímos uma *timeline* de 4 linhas, pois agora a cardinalidade de L apresenta o seguinte resultado: $|L| = 4$. Neste caso os eventos estão melhor organizados, ocupando menos linhas na *timeline*.

3. Trabalhos relacionados

Os trabalhos apresentados nesta seção serviram de base para o projeto, e sabemos que o problema pode ser mapeado a coloração de grafos que é classificado como NP-Difícil, então até o momento não existe um algoritmo que resolva este 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* [12], que é uma aplicação *web* 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 a aplicação exige que toda organização deve ser feita manualmente.

Apesar do artigo de Neufeld e Tartar[2] apresentar pontos similares, não encontramos um bom objeto de comparação para este projeto, pois o foco aqui tem como objetivo organização e compactação. 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* [12] apareceu como uma ótima opção na etapa de experimentação.

3.1. 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 [1]”. O autor 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 [1] é uma versão mais simples do 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 por Garey e Johnson [3] como NP-Difícil.

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

Este artigo que foi publicado na ACM (*Association for Computing Machinery*), falando sobre o problema do cronograma do professor. No artigo de Neufeld e Tartar [2], 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.

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

Neste artigo Welsh e Powell [4] 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 grafo, 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.

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 vértices 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.

Algorithm 1 Largest-First(Grafo)

```
1: Grafo.vértices.ordenaPorGrau()           ▷ ordenação pela quantidade de arestas
2: listaCores  $\leftarrow$  [Grafo.vértices[0]]      ▷ Lista das Cores
3: Grafo.vértices.remove(0)                  ▷ Remove o primeiro elemento
4: para cada i em Grafo.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 vértice 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)
```

3.4. Um algoritmo heurístico aplicado ao problema de corte unidimensional

Este artigo Landes [5] demonstra a utilização de meta heurística *Greedy Randomized Adaptive Search Procedure (GRASP)* para gerar uma solução inicial de boa qualidade e a meta-heurística *Iterated Local Search (ILS)* como busca local.

A partir disso o autor utiliza o Método Randômico Não Ascendente (MRNA) [5] como uma heurística de refinamento, esta técnica se comporta como um algoritmo genético. O MRNA consiste em escolher aleatoriamente uma solução vizinha à solução corrente, aceitando esta vizinha somente se o número de conflitos não for aumentado. O procedimento termina depois que um determinado número de iterações sem melhora (*maxit*), passado como parâmetro, é alcançado.

Algorithm 2 MRNA(G, s, maxit)

```
1: contador  $\leftarrow 0$  ▷ cria contador
2: enquanto contador < maxit em
3:    $s' \leftarrow \text{random}(s)$ 
4:   se  $f(s) < f'(s)$  : então
5:     contador  $\leftarrow 0$ 
6:   senão
7:     contador  $\leftarrow \text{contador} + 1$ 
8:   fim se
9:    $s \leftarrow s'$ 
10: fim enquanto
11: devolve RNA
```

3.5. New methods to color the vertices of a graph

O algoritmo proposto por Brélaz [7], também é um algoritmo com caráter guloso, sendo junto do *LF* uma das principais heurísticas para resolver este problema. Este algoritmo é exato para grafos bipartidos e importante para buscar cliques maximais em grafos, com 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.

3.6. chromatic scheduling and the chromatic number problem.

Este artigo escrito por Brown [6] 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 os dados do problema crescem torna-se impossível que o algoritmo termine de rodar.

4. Algoritmo

A implementação do algoritmo foi construída em *Python* [13] e pode ser dividida em 3 etapas que são: a entrada e o tratamento de dados, organização dos eventos e por último

o *plot* do gráfico. Antes de apresentar a construção do algoritmo devemos compreender sob qual hipótese o algoritmo foi construído.

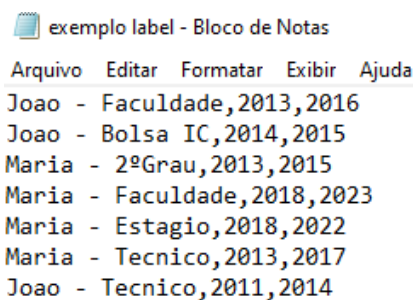
4.1. Hipótese

O algoritmo proposto organiza eventos em um gráfico e foi construído partindo da seguinte hipótese: os eventos que cobrem um grande intervalo de tempo possuem mais chances de ter conflito com outros eventos, pois ocuparão uma parte maior de uma linha no gráfico. Então os eventos com maior duração devem ser posicionados primeiro e os menores devem se adaptar buscando os lugares restantes.

Uma característica interessante desta hipótese é que ela pode ser testada de maneira inversa, ou seja, tentando colocar os menores eventos primeiro e comparando os resultados.

4.2. 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 a um evento diferente, que possui os seguintes atributos: nome, data de início e data final. Os atributos de um evento são separados por vírgula, como demonstra a Figura 3.



```
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 3. Entrada de dados com label

O primeiro passo do algoritmo é criar uma lista e inserir os eventos enquanto ocorre a leitura do documento de texto, cada linha deste arquivo será um evento que possui seus atributos separados por vírgula. A fim de aproveitar a iteração sobre a lista, um novo atributo que não está presente no arquivo de entrada é criado em execução. Este novo atributo é o tamanho do intervalo de cada evento, que é dado por $intervalo = data_{final} - data_{inicial}$, este atributo ajudará durante a ordenação dos eventos.

Durante a criação da lista de eventos são guardadas duas datas para definir os limites no eixo x do *plot* do gráfico. Chamaremos a primeira data de *first*, pois ela guardará a data inicial do evento que começou primeiro, e a segunda data será *last*, pois guardará a data final do evento que terminou por ultimo.

4.3. Organização dos eventos

O *Timeliner* se diferencia do algoritmo *Largest-First* pois este calcula o grau de cada um dos vértices do grafo para a ordenação, que depende do grau dos vértices. O *Timeliner* também precisa ordenar os eventos, mas como o seu critério não é o grau e sim a duração dos eventos, não é necessário calcular o grau, diminuindo a quantidade de operações para ordenação ao se comparar com o *Largest-First*.

Algorithm 3 Timeliner(eventos)

```
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:     verificaSobreposição  $\leftarrow$  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 vértice 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 a etapa da ordenação, é criada uma lista de linhas para o *plot*, e a sua primeira linha contém o primeiro evento da lista de eventos, que devido a ordenação é o evento com a maior duração. Este evento é removido da lista de eventos a serem posicionados.

Então é necessário criar uma lista de linhas, para agrupar os eventos que estão na mesma linha. Assim começa o processo de definir a posição de cada evento no gráfico. Para cada evento é necessário verificar se este possui conflito de data com algum evento da linha 1, caso não haja conflito, este evento será adicionado na primeira linha. Se existir conflito, repetiremos o processo para as linhas seguintes. Caso o evento tenha um conflito de data em todas as linhas, deverá ser adicionada na lista de linhas uma nova linha contendo este evento. Esse processo deve ser feito para todos os eventos.

4.4. Plot da linha do tempo

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



Figura 4. Linha do tempo com Label

Ao observar a Figura 4 podemos observar que o algoritmo distribui algumas linhas acima eixo x, e outras estão distribuídas abaixo. O algoritmo intercala as linhas no eixo x com o objetivo de facilitar a visualização das datas, deixando todos os eventos o mais próximo possível das datas. Diferente da Figura 1, pois os eventos que ficavam nas linhas mais altas ficavam muito distantes de suas datas.

5. Redução entre problemas

Esta seção irá mostrar a redução do problema do número cromático para o problema da *timeline*, mapeando os eventos e linhas para vértices e arestas. Esta redução será feita com o objetivo de demonstrar que o problema tratado neste artigo é NP-Difícil.

Para compreender o problema é necessário saber que um grafo pode ser representado como $G = (V, A)$, dada essa representação temos que um grafo é formado por dois conjuntos que são conhecidos como vértices e arestas. Os vértices são representados por círculos como na Figura 5, e as arestas são as linhas que ligam os vértices. A partir disso podemos investigar o problema da coloração de grafos, ou problema do número cromático, que é um dos 21 problemas de Karp [1] e tem como objetivo rotular os vértices de um grafo.

5.1. Coloração de Grafos

Um grafo colorido corretamente, conforme o problema do número cromático, deve satisfazer a regra de que dois vértices adjacentes não podem compartilhar a mesma cor, ou seja, dois vértices ligados por uma aresta devem possuir cores (ou rótulos) diferentes. 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.

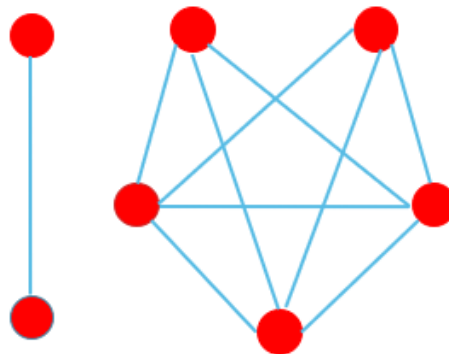


Figura 5. Grafo sem coloração

A Figura 5 representa exatamente o mesmo grafo que a Figura 6. A diferença entre as duas imagens deve-se a coloração de grafos, pois a Figura 6 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 que serão formalizadas na seção a seguir:

- 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.

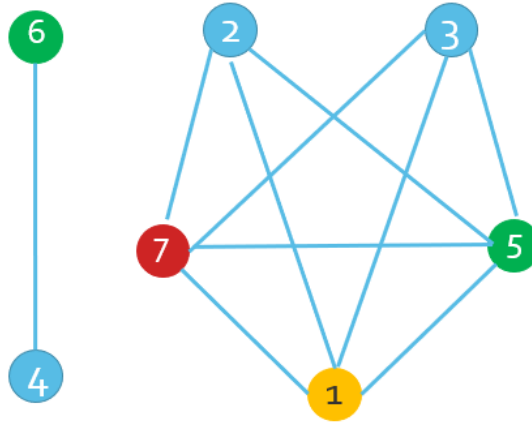


Figura 6. Coloração de grafos

5.2. Formalização

Para formalizar a redução de um problema de número cromático para o problema das timelines sem sobreposição de eventos, podemos assumir que:

- Para o problema do número cromático, temos um grafo $G = (V, A)$ dado por n vértices e m arestas.
- Para o problema do timeline, criamos um problema de timeline $T = (E, S)$ onde:
 - 1) Se $V = \{v_1, v_2, v_3, \dots, v_n\}$ então $E = \{e_1, e_2, e_3, \dots, e_n\}$
 - 2) se (v_i, v_j) pertence a A , então (e_i, e_j) pertence a S .

Agora vamos descrever a definição do grafo apresentado na Figura 6, para demonstrar que ele é equivalente a *timeline* da Figura 7. Primeiramente precisamos descrever o conjunto dos vértices, que iremos definir como $V = \{1, 2, 3, 4, 5, 6, 7\}$. Após isso é necessário representar o conjunto das arestas, que são as ligações entre vértices e podemos descrever como $A = \{(1, 2), (1, 3), (1, 5), (1, 7), (2, 5), (2, 7), (3, 5), (3, 7), (5, 7), (6, 4), (4, 5)\}$.



Figura 7. Linha do tempo

A Figura 7 é uma imagem gerada pelo *Timeliner*, para escrever sua definição é necessário lembrar que uma *timeline* é representada por $T = (E, S)$, e te-

mos que $E = \{1, 2, 3, 4, 5, 6, 7\}$ ao definir o conjunto dos eventos. Sabendo que cada sobreposição ocorre quando eventos compartilham datas, temos que $S = \{(1, 2), (1, 3), (1, 5), (1, 7), (2, 5), (2, 7), (3, 5), (3, 7), (5, 7), (6, 4)\}$.

Neste momento já é possível visualizar o grafo da Figura 6 como a *timeline* da Figura 7, pois o conjunto dos vértices possui os mesmos elementos que o conjunto de eventos, além disso, as arestas que ligam os eventos são equivalentes às sobreposições de eventos. O próximo passo é compreender a redução dos problemas, para isso é necessário analisar as restrições de cada problema. O problema de coloração de grafos não permite que o conjunto de cores possua dois vértices que são ligados por uma aresta, e o problema da *timeline* não permite que um conjunto de linhas possua dois eventos com sobreposição de datas.

Grafo		Timelines
Vértices	1, 2, 3, 4, 5, 6, 7	Eventos
Arestas	(1,2),(1,3),(1,5),(1,7),(2,5),(2,7),(3,5),(3,7),(5,7),(6,4)	Intersecções
Cores	$\{1\}, \{2,3,4\}, \{5,6\}, \{7\}$	Linhas
$ C $	4	$ L $

Como já estabelecemos a ligação entre arestas e sobreposições, o que resta é visualizarmos que o conjunto das linhas é igual ao conjunto das cores. Então podemos ver que $linha1 = \{1\}$, $linha2 = \{2, 3, 4\}$, $linha3 = \{5, 6\}$ e $linha4 = \{7\}$, a partir disso o conjunto das linhas está definido como $L = \{\{1\}, \{2, 3, 4\}, \{5, 6\}, \{7\}\}$, ao analisar a Figura 6 vemos que os eventos em uma linha são os mesmo vértices coloridos por uma cor. 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 está sendo resolvido.

6. Resultados

Um dos objetivos desta seção é a validação da hipótese que foi fundamental para a construção do algoritmo. Esta hipótese é descrita na Seção 4.1, e parte da suposição que os eventos que cobrem um grande intervalo de tempo possuem mais chances de ter conflito, pois ocupam uma parte maior de uma linha no gráfico.

Após o teste da hipótese o objetivo torna-se colocar a melhor versão do *Timeliner* em competição com outras heurísticas que resolvem o problema da coloração de grafos, pois queremos compreender se o algoritmo cumpre sua proposta, que é disponibilizar uma boa organização de eventos de forma rápida. Estas competições irão nivelar os resultados encontrados pelo *Timeliner*, portanto o nosso objetivo é comparar o número de linhas obtido pelo *Timeliner*, com o número de cores encontrados pelas demais heurísticas. Caso os resultados sejam parecidos, o desempate será caracterizado pelo número de comparações feito por cada algoritmo, as comparações contabilizadas são efetuadas durante a inserção de um evento no *Timeliner*. Para implementar as outras heurísticas utilizaremos a ferramenta *Sagemath* [11] com o objetivo de obter os resultados, pois a ferramenta proporciona a implementação de um grafo de forma rápida.

6.1. Validação da Hipótese

O objetivo desta seção é comparar os resultados obtidos ao aplicar o algoritmo *Timeliner* comparando com uma técnica que vai contra esta hipótese. Vale ressaltar que a base do *Timeliner* é ordenar os eventos pelo tamanho do intervalo de forma decrescente, isto é, os eventos com intervalos maiores terão prioridade.

Para esta validação a heurística da proposta original será comparada com uma heurística inversa, que faz a ordenação de forma crescente. O intuito desta comparação é salientar que se a hipótese proposta traz vantagens, então sua inversa deve trazer desvantagem, destacando as diferenças. O intuito aqui é descobrir qual das duas técnicas permite a organização mais compacta de uma *timeline*, ou seja, queremos descobrir qual algoritmo obtém a menor quantidade de linhas de forma mais eficiente, que neste caso significa utilizar poucas comparações para encontrar o resultado.

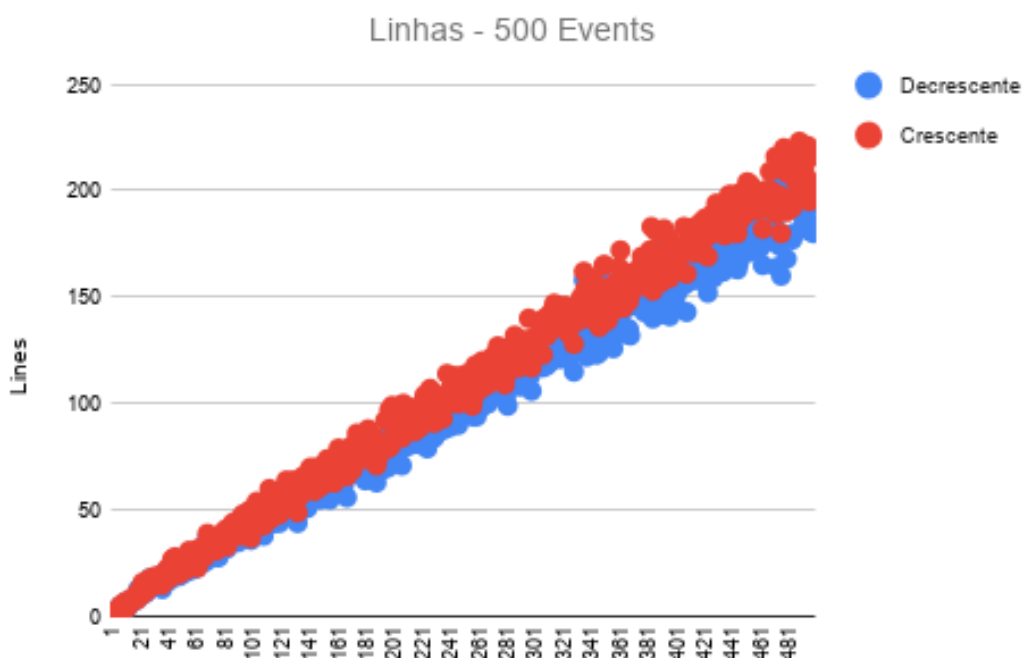


Figura 8. Linhas - validação da hipótese

Para esta comparação foi criada uma amostra de 500 *timelines* geradas aleatoriamente, que podem ser encontrados no github do projeto. Todas as 500 *timelines* estão representadas em um arquivo de texto, possuindo de 1 até 500 eventos. As duas versões do algoritmo *Timeliner* rodaram essa mesma base de dados, e os resultados pode ser visualizados na Figura 8. Para compreender o resultado gerado precisamos lembrar que a heurística ideal é aquela que apresenta um menor número de linhas, como o eixo y do gráfico esta representando as linhas, a heurística mais eficiente é aquela que estiver próxima do eixo x mais vezes.

A heurística decrescente, que é baseada na hipótese levantada por este artigo, está representada na Figura 8 pelos pontos azuis, e os pontos vermelhos representam a heurística crescente, que funciona de forma oposta. A heurística decrescente apresentou

os melhores resultados nos testes, pois encontrou mais vezes uma quantidade menor de linhas, estando mais vezes mais próxima do eixo x. Para ser mais exato, a heurística crescente encontrou mais linhas em 473 casos, a decrescente encontrou mais linhas uma única vez e nos 26 casos restantes as heurísticas obtiveram o mesmo resultado.

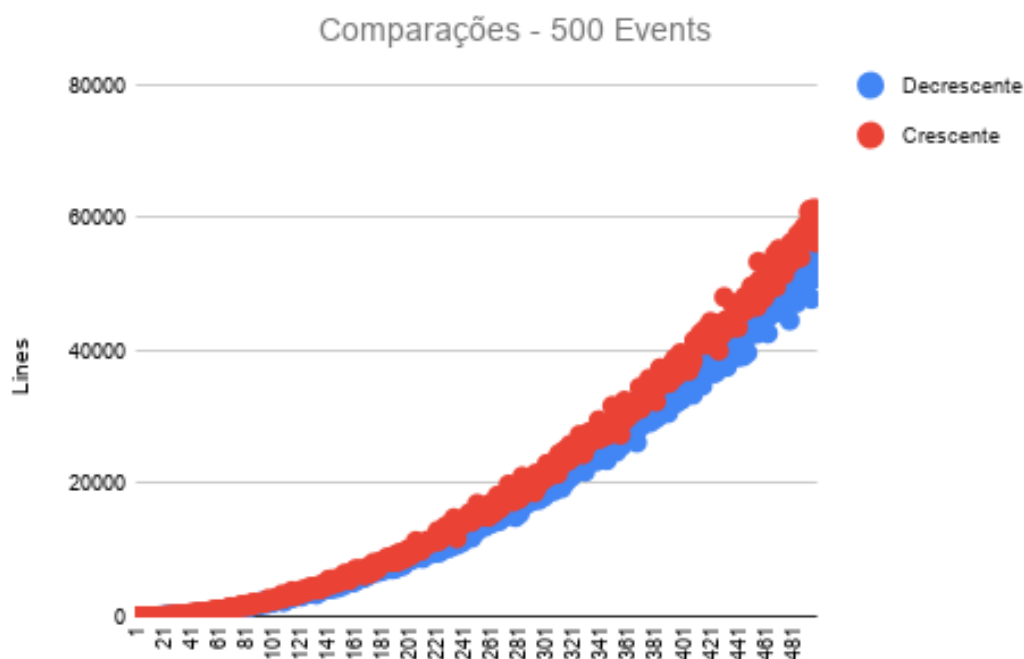


Figura 9. Comparações - validação da hipótese

Além do algoritmo baseado na hipótese original construir *timelines* com menos linhas, podemos ver na Figura 9 que precisa de menos comparações para encontrar um resultado melhor. Para ser mais exato, a heurística crescente precisou de mais comparações em 461 casos, a decrescente 33 vezes e nos 6 casos restantes as heurísticas precisaram da mesma quantidade de comparações. Ao analisar o resultados referentes a quantidade de linhas e quantidade de comparações, podemos concluir que a hipótese original se mostrou verdadeira nesta base de dados.

6.2. Heurísticas

Nesta seção a heurística proposta será comparada com outras heurísticas apresentadas na Seção 3, referentes ao problema do número cromático. Como estamos operando sobre grafos, as duas heurísticas apresentadas a seguir tiveram sua implementação feita no *software SageMath* [11], pois a ferramenta já possui bibliotecas para grafos e suas funções, isso permite trabalhar com grafos de forma rápida.

6.2.1. Largest-First

A primeira das heurísticas é a LF (Largest-First) [4], que foi criada por Welsh e Powell e está descrita na Seção 3.3. Como a finalidade desta heurística é resolver o problema do número cromático não é necessário fazer nenhuma modificação ao implementar este

algoritmo no *SageMath*. O LF pode ser dividido em 3 etapas que são: descobrir o grau de cada vértice, ordenar os vértices de forma decrescente pelo grau e colorir os vértices. O *Timeliner* também pode ser dividido da seguinte forma: descobrir o intervalo de cada evento, ordenar os eventos de forma decrescente pelo intervalo e inserir os eventos.

Analisando estas etapas é possível enxergar semelhanças entre as heurísticas, mas já podemos afirmar que a heurística proposta neste trabalho vai apresentar um número menor de comparações, pois para descobrir o grau dos vértices é necessário comparar todos os vértices do grafo, enquanto o *Timeliner* precisa descobrir o tamanho do intervalo, que é dado por uma simples operação de subtração. Contudo, ainda é necessário descobrir o quanto esta quantidade de comparações impacta no resultado final.

6.2.2. MRNA

As comparações não estão restritas à abordagem gulosa, pois também utilizaremos uma versão modificada da heurística MRNA [5] que é um algoritmo genético apresentado na Seção 3.4. Este algoritmo não foi criado para a finalidade a qual estamos utilizando, sua descrição é genérica para resolver problemas de forma genética, devido a isto foram feitas algumas adaptações. O MRNA implementado no *SageMath* recebe uma lista de cores com um vértice para cada cor, o objetivo aqui é gerar uma nova população com os vértices agrupados em uma quantidade menor de cores. Para esta adaptação do MRNA, descrevemos uma nova população como uma nova lista de cores, esta lista é construída a partir de um movimento aleatório que remove um vértice de um grupo de cores e associa este vértice a uma outra cor.

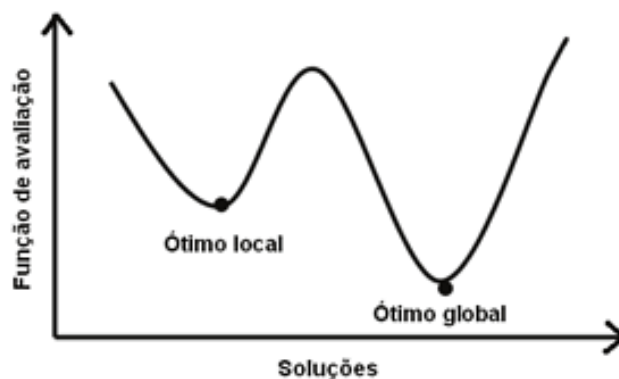


Figura 10. Ótimo local

A segunda modificação foi feita pois o algoritmo ia ficar preso no primeira solução que representasse um ótimo local, se usássemos a versão original. Isso aconteceria pois o MRNA só salva uma nova população se essa apresenta um resultado melhor, ou seja, ficaríamos presos no primeiro ponto da Figura 10. Isso aconteceria pois ao não admitir movimentos que formam populações piores, não seria possível chegar no segundo ponto da Figura 10, que é um ponto de ótimo global, porque neste caso é necessário subir no gráfico (piorar), para descer até o ótimo global.

Para realizar esta adaptação, criamos duas listas auxiliares, uma lista para guardar

a melhor população encontrada até o momento, e a outra lista servirá para guardar a ultima população. A lista que guarda a ultima população é atualizada a cada iteração, e a lista da melhor solução só é atualizada quando a ultima população gerada obtiver uma quantidade de cores inferior que a melhor solução. Os testes gerados na seção a seguir utilizam $maxit = 100$ e para cada *timeline* o MRNA é chamado 50 vezes, garantindo que a quantidade de iterações seja sempre superior a 500.

6.2.3. Comparação das heurísticas

O objetivo desta seção é colocar o *Timeliner* em competição com outras heurísticas, sendo uma delas o *LF* que possui um caráter guloso, como o nosso algoritmo, e o MRNA que possui caráter genético. O intuito aqui é descobrir qual das técnicas permite a organização mais compacta, ou seja, queremos descobrir qual algoritmo obtém a menor quantidade de linhas ou cores de forma mais eficiente, que neste caso significa utilizar menos comparações para encontrar o resultado.

Para esta comparação foi criada uma amostra de 500 *timelines* geradas aleatoriamente, que podem ser encontrados no github do projeto. Todas as 500 *timelines* estão representadas em um arquivo de texto, possuindo de 1 até 500 eventos. As três heurísticas rodaram essa mesma base de dados, e os resultados pode ser visualizados na Figura 11. Para compreender o resultado gerado precisamos lembrar que um bom resultado significa apresentar um menor número de linhas, como o eixo y do gráfico esta representando as linhas, a heurística mais eficiente é aquela que estiver próxima do eixo x mais vezes.

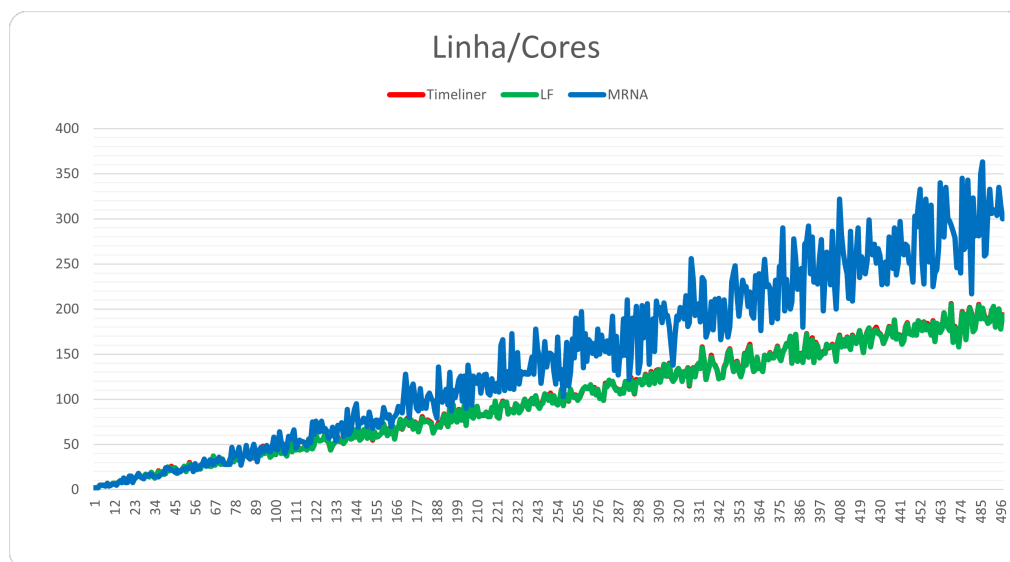


Figura 11. Cores/linhas

Ao analisar os resultados da Figura 11 podemos observar que MRNA foi a heurística que mais se distanciou do eixo x, mas vale ressaltar que nos casos menores esta heurística obteve resultados próximos das demais heurísticas. Este caso é o mais complexo de avaliar porque é possível que ao aumentar o $maxit$ para os casos maiores, seria possível encontrar resultados melhores, mas não podemos afirmar qual deveria ser valor de $maxit$, pois o algoritmo é genético, funcionando de forma aleatória.

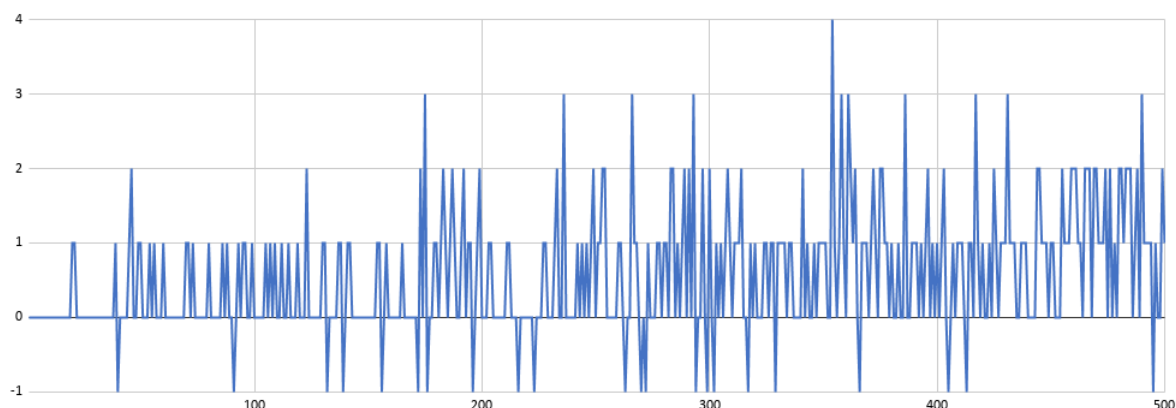


Figura 12. *Timeliner*(Cores) - *LF*(Linhas)

O *Timeliner* não foi a heurística que mais vezes encontrou o melhor resultado, mas esteve tão próximo do *LF* que fica difícil enxergá-los na Figura 11, apenas é possível visualizar o *Timeliner* pois existem alguns pontos vermelhos acima dos pontos verdes. Devido a dificuldade de visualizar o *Timeliner* na Figura 11 criamos outro gráfico para facilitar a visualização dos resultados. Na Figura 12 é apresentado o resultado ao aplicar a subtração entre os resultados do *Timeliner* e os resultados do *LF*. Todos os resultados que estão acima do eixo x representam as vezes que o *LF* obteve um melhor desempenho, já os resultados abaixo do eixo x representam as vezes que o *Timeliner* se saiu melhor. Podemos notar que o *LF* ganhou mais vezes, e a maior diferença foi por 4 unidades de cores/linhas, que ocorreu próximos dos 350 eventos.

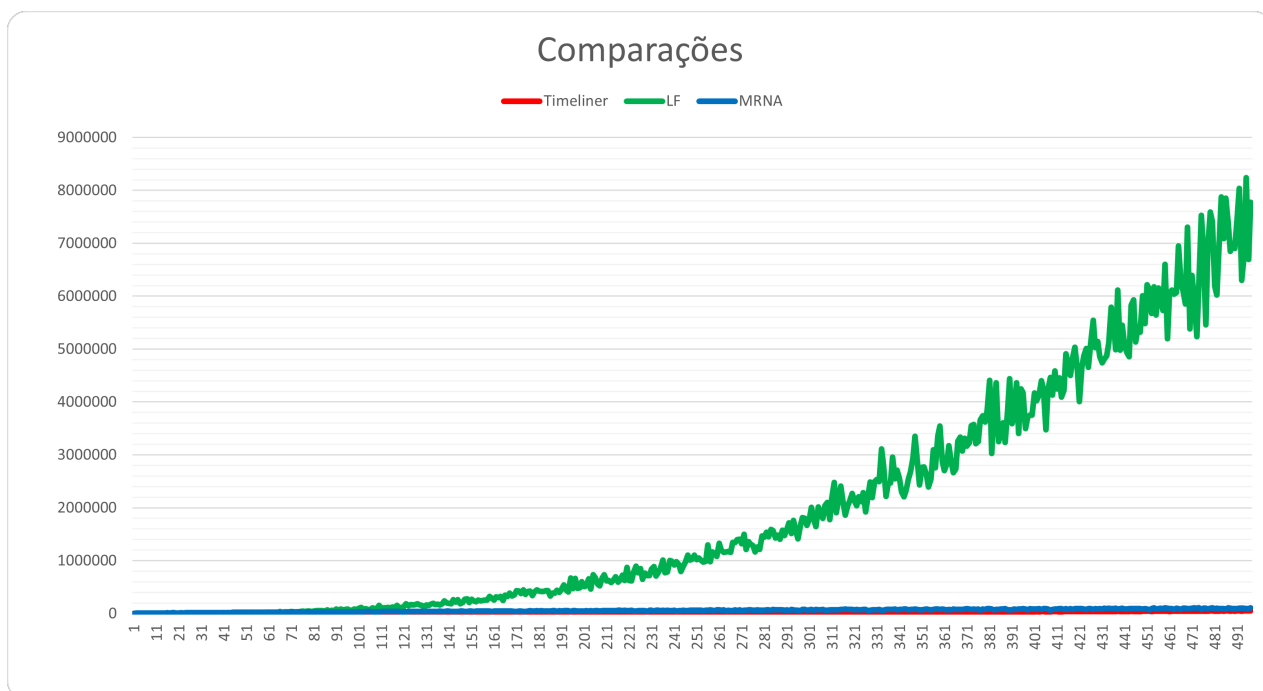


Figura 13. Comparações

A Figura 13 deixa evidente a relação de eficiência que buscamos ao construir o

Timeliner, pois mesmo que o *LF* encontre mais vezes uma quantidade inferior de cores, o preço da quantidade de comparações é bem mais alta ao se comparar com as demais heurísticas. Acreditamos que se no MRNA o valor do *maxit* fosse mais alto, poderíamos chegar em uma quantidade de comparações próximas ao *LF*, e isso reduziria a quantidade de cores do MRNA do gráfico anterior, mas o objetivo aqui é encontrar uma relação custo-benefício, então o *maxit* foi projeto para gerar um número de comparações similar ao nosso algoritmo.

Podemos perceber que o *Timeliner* encontra um resultado de linhas/cores muito próximo do *LF*, mas diferença entre a quantidade de comparações é gigante, pois a heurística *LF* precisa de muitas verificações, pelo fato de precisar encontrar o grau dos vértices. Ao analisar estes resultados, podemos concluir que o *Timeliner* se mostrou mais eficiente nesta base de dados.

7. Considerações Finais

A partir da redução apresentada na Seção 5 foi possível investigar a eficiência de nossa heurística, pois já são conhecidas muitas heurísticas eficientes para resolver o problema de coloração de grafos, além disso, esta redução permitiu mostrar que estamos lidando com um problema NP-Difícil.

Devemos levar em consideração o fato que as heurísticas comparadas resolvem todos os problemas de coloração de grafos, já o *Timeliner* não conseguiria resolver problemas de coloração de grafos com outros domínios. O problema apresentado neste artigo tem como objetivo proporcionar uma experiência rápida para quem for construir linhas do tempo. Apesar de existirem outros problemas que a quantidade de cores precisa ser a mais exata possível, devemos entender que este não é o nosso caso, então foi preciso encontrar um resultado balanceado ao pensar na relação entre a quantidade de linhas (ou cores) e comparações. Ao pensar nesta relação de eficiência podemos concluir que entre as heurísticas comparadas a *Timeliner* é melhor solução para resolver o problema das linhas do tempo.

Durante esta pesquisa existiram alguns pontos interessantes a serem ressaltados. O primeiro ponto está relacionada com a hipótese levantada para a construção deste algoritmo, pois ela demonstrou ser mais eficiente do que a proposta inversa que foi comparada na seção 6.1. Apesar de termos levantado esta hipótese e ela apresentar resultados positivos em nossa base de dados gerada aleatoriamente, não sabemos justificar o comportamento deste resultado. O segundo ponto que nos despertou a atenção durante a pesquisa foi o comportamento do MRNA adaptado por nós. O comportamento do MRNA foi muito positivo ao considerar a simplicidade deste algoritmo genético, seria muito interessante investigar o comportamento deste algoritmo nos casos mais robustos fosse aumentado o valor de parâmetro *maxit*.

Referências

- [1] Karp, R. M. (1972). Reducibility among combinatorial problems. *The Computer Journal*, 10:85–86.
- [2] 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.

- [3] Garey, M. R.; Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, ISBN 0-7167-1045-5
- [4] 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.
- [5] Landes, F.B. Semaan, G.S.S. Penna, P.H.V. (2017). Um algoritmo heurístico aplicado ao problema de corte unidimensional. XLIX Simpósio Brasileiro de Pesquisa Operacional.
- [6] Brown, J. R. (1972). Chromatic scheduling and the chromatic number problem. Management Science, 19:456–463.
- [7] Brélaz, D. (1979). New methods to color the vertices of a graph. Communications of the ACM, 22.
- [8] Lucet, C., Mendes, F., and Moukrim, A. (2006). An exact method for graph coloring. Computers Operations Research, 33(8):2189 – 2207.
- [9] Schindl, D. (2003). Graph coloring and linear programming. Presentation at First Joint Operations Research Days, Ecole Polytechnique Fédérale de Lausanne (EPFL), available on line (last visited June 2005).
- [10] de Aguiar, F. N., de Sá Carvalho Honorato, G., dos Santos, H. G., Ochi, L. S. (2005). Metaheurística busca tabu para o problema de coloração de grafos. Anais do XXXVII SBPO, pages 2497–2504.
- [11] SageMath. Open-Source Mathematical Software System, c2020. Página inicial. Disponível em: <<https://www.sagemath.org/>>. Acesso em: 20 de 11. de 2005.
- [12] Time.Graphics. Criação de uma linha do tempo - um serviço online gratuito. Página inicial. Disponível em: <<https://time.graphics/pt/>>. Acesso em: 20 de 11. de 2020.
- [13] Python. Welcome to Python.org, c2001. Página inicial. Disponível em: <<https://www.python.org/>>. Acesso em: 20 de 11. de 2020.