

Relatório Final

Maria Eduarda Bicalho

19 de Abril de 2020

1 Descrição do Problema

O projeto Maximin Share tem como objetivo fazer a divisão mais justa possível de um número de objetos com diferentes valores entre um número diferente de pessoas. O problema principal se centra em como realizar essa divisão. Dessa forma, três diferentes técnicas - Heurística, Busca Local e Busca Global - foram utilizadas para produzir três diferentes algoritmos para executar essa partição. Neste relatório essas três implementações serão analisadas com diferentes entradas para avaliar as alterações em suas saídas. Essas entradas possuirão diferentes tamanhos, alterando significativamente. Primeiramente, em relação a quantidade de pessoas e depois a quantidade de objetos. Dentro de cada uma dessas análises, serão estudados o tempo, e a qualidade da solução em relação aos diferentes dimensões de entradas. A qualidade será analisada a partir do MMS (o valor da pessoa com o menor valor), ou seja quanto maior o MMS maior a qualidade da saída.

1.1 Máquina utilizada

Memória : 1.5 GiB - Disco : 63 GB - Processador : Intel i5 - Virtualização: KVM -

2 Efeito número de pessoas

Nessa seção será realizada a análise do impacto de uma entrada com diferentes números de pessoas nos 3 diferentes algoritmos implementados no projeto. A capacidade da máquina foi sendo testada, aumentando cada vez mais o input de com o número pessoas. A variação do número de objetos foi baixa, para que a mudança dessa variável não fosse importante para o tempo e qualidade da solução, e esses se mantivessem em função do delta pessoas. A busca global não conseguiu um tempo factível depois de um número de entrada de 10 pessoas, dessa forma, esse algoritmo aparecerá somente até o valor de entrada 10 - para as demais entradas foi assumido um valor limite.

2.1 Tempo

Nesta seção, a medida utilizada para a análise será a do tempo.

```
-----FileNo
Traceback (most recent call last)<ipython-input-1-221a47c1d4ae> in
<module>
      1 for arg in args:
----> 2     heuristica.append(roda('./heuristica', arg)[1])
```

```

3         with open("out.txt",'r') as f:
4             line = f.readlines()
5             outsh.append(int(line[0]))
<ipython-input-1-13f261975a38> in roda(ex, in_f)
6         with open(in_f) as f:
7             start = time.perf_counter()
----> 8             proc = subprocess.run([ex], input=f.read(), text=True,
capture_output=True)
9             end = time.perf_counter()
10            return proc.stdout, end-start
/usr/lib/python3.8/subprocess.py in run(input, capture_output,
timeout, check, *popenargs, **kwargs)
487            kwargs['stderr'] = PIPE
488
--> 489        with Popen(*popenargs, **kwargs) as process:
490            try:
491                stdout, stderr = process.communicate(input,
timeout=timeout)
/usr/lib/python3.8/subprocess.py in __init__(self, args, bufsize,
executable, stdin, stdout, stderr, preexec_fn, close_fds, shell, cwd,
env, universal_newlines, startupinfo, creationflags, restore_signals,
start_new_session, pass_fds, encoding, errors, text)
852                encoding=encoding, errors=errors)
853
--> 854            self._execute_child(args, executable, preexec_fn,
close_fds,
855                                pass_fds, cwd, env,
856                                startupinfo, creationflags,
shell,
/usr/lib/python3.8/subprocess.py in _execute_child(self, args,
executable, preexec_fn, close_fds, pass_fds, cwd, env, startupinfo,
creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
errwrite, restore_signals, start_new_session)
1700            if errno_num != 0:
1701                err_msg = os.strerror(errno_num)
-> 1702                raise child_exception_type(errno_num,
err_msg, err_filename)
1703            raise child_exception_type(err_msg)
1704
FileNotFoundError: [Errno 2] No such file or directory: './heuristica'

```

-----ValueE

```

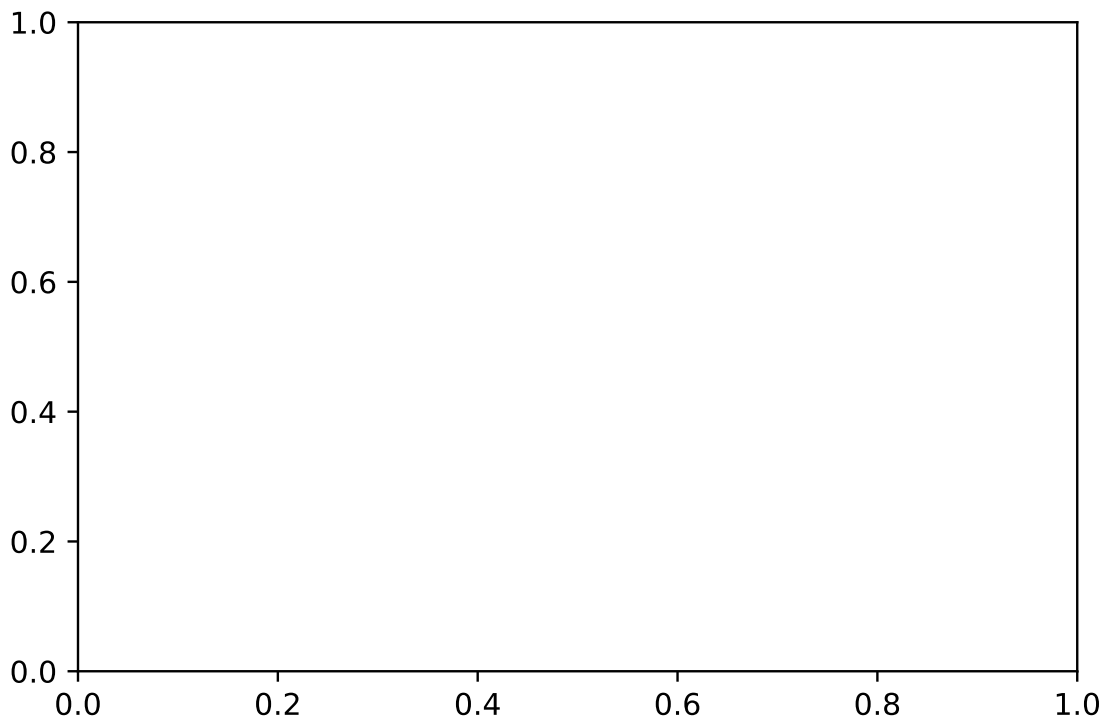
Traceback (most recent call last)<ipython-input-1-77b91d3608d6> in
<module>
2
3 x= range(1,20,2)
----> 4 h,= plt.plot(x,heuristica[:10], 'b' ,label = "Heurística")
5 l,= plt.plot(x,local[:10], 'm' ,label = 'Busca Local')
6 g, = plt.plot(x,exaus[:10], 'g',label = 'Busca Global')
~/lib/python3.8/site-packages/matplotlib/pyplot.py in
plot(scalex, scaley, data, *args, **kwargs)
3017 @_copy_docstring_and_deprecators(Axes.plot)
3018 def plot(*args, scalex=True, scaley=True, data=None,

```

```

**kwargs):
-> 3019     return gca().plot(
      3020         *args, scalex=scalex, scaley=scaley,
      3021         **({"data": data} if data is not None else {}),
**kwargs)
~/.local/lib/python3.8/site-packages/matplotlib/axes/_axes.py in
plot(self, scalex, scaley, data, *args, **kwargs)
      1603         """
      1604         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1605         lines = [*self._get_lines(*args, data=data, **kwargs)]
      1606         for line in lines:
      1607             self.add_line(line)
~/.local/lib/python3.8/site-packages/matplotlib/axes/_base.py in
__call__(self, data, *args, **kwargs)
      313             this += args[0],
      314             args = args[1:]
--> 315             yield from self._plot_args(this, kwargs)
      316
      317     def get_next_color(self):
~/.local/lib/python3.8/site-packages/matplotlib/axes/_base.py in
_plot_args(self, tup, kwargs, return_kwargs)
      499
      500         if x.shape[0] != y.shape[0]:
--> 501             raise ValueError(f"x and y must have same first
dimension, but "
      502                             f"have shapes {x.shape} and
{y.shape}")
      503         if x.ndim > 2 or y.ndim > 2:
ValueError: x and y must have same first dimension, but have shapes
(10,) and (0,)

```



A partir dos gráficos pode-se concluir que os algoritmos de busca local de busca global possuem valores de tempo maiores do que a heurística, a qual não sofre fortes alterações. Um fato curioso que pode ser observado é que a busca local inicia com um tempo maior e logo antes da entrada de 10 - por volta da entrada de 7- é ultrapassada rapidamente pela busca global, cuja variação é extremamente alta. Por realizar a busca no resultado apresentado e não em todos os possíveis, a busca local já possui um tempo maior, mas factível na máquina utilizada e possi um resultado melhor que o da heurística. Por ser um algoritmo que utiliza recursão e percorre todos os caminhos possíveis para certificar que possui o melhor, a implementação da busca global cresce rapidamente. O único esforço da solução heurística é o de ordenar, por isso o tempo não sofre grandes alterações.

2.2 Qualidade da solução

```
-----IndexE
Traceback (most recent call last)<ipython-input-1-8b5e1bf85297> in
<module>
      3 x= range(1,sz*2,2)
      4 for i in range(sz-5):
----> 5     outsg.append(outsg[-1])
      6
      7 h,= plt.plot(x,outsh,'b' ,label = "Heurística")
IndexError: list index out of range
```

A análise da qualidade da solução com um aumento significativo no número de pessoas com uma mudança pequena na quantidade de objetos não é muito eficiente, pois em determinado valor, não terá objetos suficientes para a quantidade de pessoas. Dessa forma somente entradas bem pequenas são avaliadas. Todavia, mesmo com essa restrição já é possível analisar que o resultado da busca global e da busca local possui um maior qualidade do que o da heurística.

3 Efeito número de objetos

Nessa seção será realizada a análise do impacto de uma entrada com diferentes números de objetos nos 3 diferentes algoritmos implementados no projeto. A capacidade da máquina foi sendo testada, aumentando cada vez mais o input de com o número objetos. A variação do número de pessoas foi baixa, para que a mudança dessa variável não fosse importante para o tempo e qualidade da solução, e esses se mantivessem em função do delta objetos. A busca global não conseguiu um tempo factível depois de um número de entrada de 10 pessoas. Dessa forma, esse algoritmo aparecerá somente até o valor de entrada 10 - para as demais entradas foi assumido um valor limite.

3.1 Tempo

Nesta seção, a medida utilizada para a análise será a do tempo.

```
-----FileNo
Traceback (most recent call last)<ipython-input-1-e8d8b3bdb241> in
<module>
      1 for arq in arqs:
----> 2     heuristica.append(roda('./heuristica', arq)[1])
      3     with open("out.txt",'r') as f:
```

```

4         line = f.readlines()
5         outsh.append(int(line[0]))
<ipython-input-1-13f261975a38> in roda(ex, in_f)
6         with open(in_f) as f:
7             start = time.perf_counter()
----> 8             proc = subprocess.run([ex], input=f.read(), text=True,
capture_output=True)
9             end = time.perf_counter()
10            return proc.stdout, end-start
/usr/lib/python3.8/subprocess.py in run(input, capture_output,
timeout, check, *popenargs, **kwargs)
487            kwargs['stderr'] = PIPE
488
--> 489        with Popen(*popenargs, **kwargs) as process:
490            try:
491                stdout, stderr = process.communicate(input,
timeout=timeout)
/usr/lib/python3.8/subprocess.py in __init__(self, args, bufsize,
executable, stdin, stdout, stderr, preexec_fn, close_fds, shell, cwd,
env, universal_newlines, startupinfo, creationflags, restore_signals,
start_new_session, pass_fds, encoding, errors, text)
852                encoding=encoding, errors=errors)
853
--> 854            self._execute_child(args, executable, preexec_fn,
close_fds,
855                                pass_fds, cwd, env,
856                                startupinfo, creationflags,
shell,
/usr/lib/python3.8/subprocess.py in _execute_child(self, args,
executable, preexec_fn, close_fds, pass_fds, cwd, env, startupinfo,
creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
errwrite, restore_signals, start_new_session)
1700                if errno_num != 0:
1701                    err_msg = os.strerror(errno_num)
-> 1702                raise child_exception_type(errno_num,
err_msg, err_filename)
1703                raise child_exception_type(err_msg)
1704
FileNotFoundError: [Errno 2] No such file or directory: './heuristica'

```

-----FileNo

Traceback (most recent call last)<ipython-input-1-aa463eb6ea47> in
<module>

```

1 for arq in arqsg:
----> 2     exaus.append(roda('./global', arq)[1])
3     with open("out.txt",'r') as f:
4         line = f.readlines()
5         outsg.append(int(line[0]))
<ipython-input-1-13f261975a38> in roda(ex, in_f)
6         with open(in_f) as f:
7             start = time.perf_counter()
----> 8             proc = subprocess.run([ex], input=f.read(), text=True,
capture_output=True)

```

```

9         end = time.perf_counter()
10        return proc.stdout, end-start
/usr/lib/python3.8/subprocess.py in run(input, capture_output,
timeout, check, *popenargs, **kwargs)
487            kwargs['stderr'] = PIPE
488
--> 489        with Popen(*popenargs, **kwargs) as process:
490            try:
491                stdout, stderr = process.communicate(input,
timeout=timeout)
/usr/lib/python3.8/subprocess.py in __init__(self, args, bufsize,
executable, stdin, stdout, stderr, preexec_fn, close_fds, shell, cwd,
env, universal_newlines, startupinfo, creationflags, restore_signals,
start_new_session, pass_fds, encoding, errors, text)
852                encoding=encoding, errors=errors)
853
--> 854            self._execute_child(args, executable, preexec_fn,
close_fds,
855                                pass_fds, cwd, env,
856                                startupinfo, creationflags,
shell,
/usr/lib/python3.8/subprocess.py in _execute_child(self, args,
executable, preexec_fn, close_fds, pass_fds, cwd, env, startupinfo,
creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
errwrite, restore_signals, start_new_session)
1700                if errno_num != 0:
1701                    err_msg = os.strerror(errno_num)
-> 1702                raise child_exception_type(errno_num,
err_msg, err_filename)
1703                raise child_exception_type(err_msg)
1704

```

FileNotFoundError: [Errno 2] No such file or directory: './global'

-----ValueError

Traceback (most recent call last)<ipython-input-1-a8b2b4d2c39e> in
<module>

```

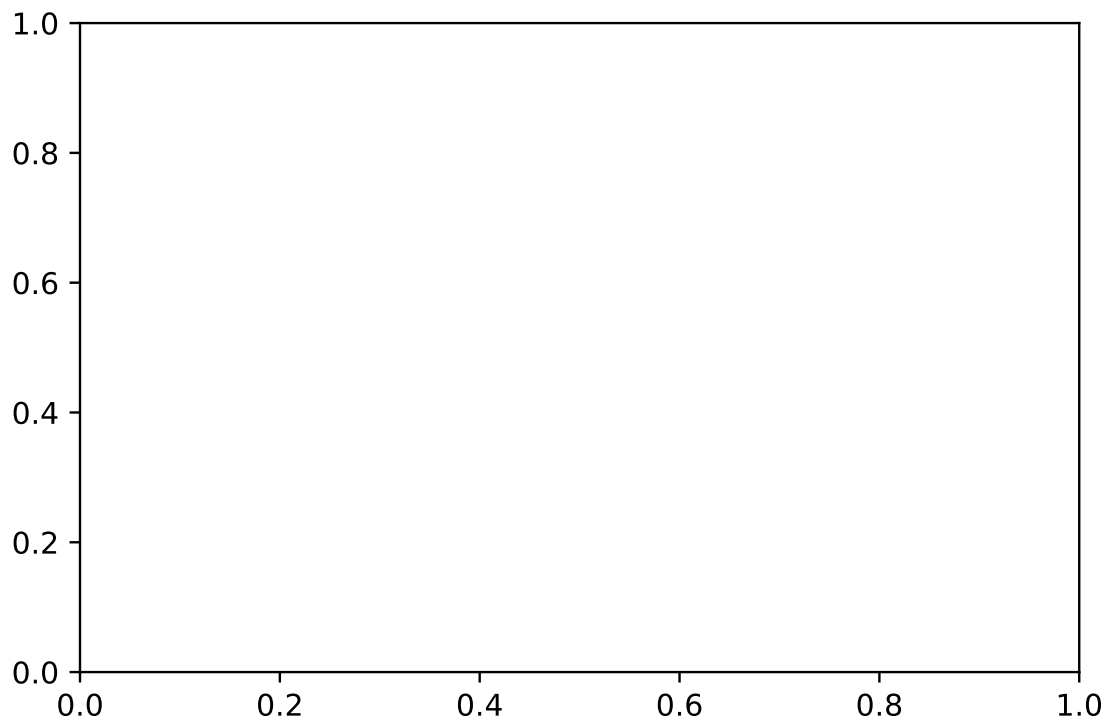
3 import matplotlib.pyplot as plt
4 x= range(1,10,2)
----> 5 h,= plt.plot(x,heuristica[:5],'b' ,label = "Heurística")
6 l,= plt.plot(x,local[:5],'m' ,label = 'Busca Local')
7 g, = plt.plot(x,exaus[:5], 'g',label = 'Busca Global')
~/lib/python3.8/site-packages/matplotlib/pyplot.py in
plot(scalex, scaley, data, *args, **kwargs)
3017 @_copy_docstring_and_deprecators(Axes.plot)
3018 def plot(*args, scalex=True, scaley=True, data=None,
**kwargs):
-> 3019     return gca().plot(
3020         *args, scalex=scalex, scaley=scaley,
3021         **({"data": data} if data is not None else {}),
**kwargs)
~/lib/python3.8/site-packages/matplotlib/axes/_axes.py in
plot(self, scalex, scaley, data, *args, **kwargs)
1603         """

```

```

1604         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1605         lines = [*self._get_lines(*args, data=data, **kwargs)]
1606         for line in lines:
1607             self.add_line(line)
~/local/lib/python3.8/site-packages/matplotlib/axes/_base.py in
__call__(self, data, *args, **kwargs)
    313             this += args[0],
    314             args = args[1:]
--> 315             yield from self._plot_args(this, kwargs)
    316
    317     def get_next_color(self):
~/local/lib/python3.8/site-packages/matplotlib/axes/_base.py in
_plot_args(self, tup, kwargs, return_kwargs)
    499
    500         if x.shape[0] != y.shape[0]:
--> 501             raise ValueError(f"x and y must have same first
dimension, but "
    502                             f"have shapes {x.shape} and
{y.shape}")
    503         if x.ndim > 2 or y.ndim > 2:
ValueError: x and y must have same first dimension, but have shapes
(5,) and (0,)

```



Como analisado no gráfico do tempo da seção anterior, a busca local e a busca global possuem uma variação significativamente maior do que a heurística. Nas entradas pequenas as 3 implementações possuem tempos parecidos, mas há uma relação direta com o aumento da quantidade de objetos na entrada e com o aumento do tempo da busca global e local, enquanto o tempo da heurística não altera muito. Novamente é perceptível a rápida mudança no tempo da busca global, que inicia inferior ao da busca local e depois aumenta drasticamente.

3.2 Qualidade da solução

3.2.1 Gráficos

```
-----IndexE
Traceback (most recent call last)<ipython-input-1-bef033elace2> in
<module>
      2 x= range(1,10,2)
      3 for i in range(sz-5):
----> 4     outsg.append(outsg[-1])
      5
      6 h,= plt.plot(x,outsh[:5], 'b' ,label = "Heurística")
IndexError: list index out of range
```

No último gráfico de qualidade da solução pode ser observado que com entradas pequenas os três algoritmos possuem resultados parecido. Contudo, com o aumento do número de objetos a diferença entre a busca local heurística vai se tornando cada vez maior. No primeiro gráfico de qualidade da solução pode ser observado que a busca global estava crescendo com variação parecida da busca local, ou seja, pode se que essas duas implementações, se possível avaliar, tivessem resultados mais parecidos do que os da busca local e da heurística.

4 Conclusão

As implementações analisadas no relatório possuem algumas características diferentes. Para entradas pequenas os algoritmos apresentam soluções com qualidades muito parecidas, e o tempo da heurística já inicia um pouco menor. Dessa forma esse algoritmo parece ser eficiente para entradas pequenas, uma vez que é mais rápido e possui saídas com alta qualidade. Na medida que o a entrada aumenta o tempo da busca local e global aumenta, e a qualidade da solução da heurística cai. Assim sendo, para entradas cada vez maiores, resultados cada vez maiores são tidos a partir da busca local. Consequentemente, as implementações podem ser utilizadas para diferentes finalidades, cada uma podendo ser escolhida a partir das preferências e importâncias de determinado projeto.