Relatório Final

Maria Eduarda Bicalho

30 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 pincipal se centra em como realizar essa divisão. Dessa forma, três diferentes técnicas - Herística, Busca Local e Busca Global - foram utilizadas para produzir três diferentes algoritmos para executar essa partição. A busca local possui 3 métodos de implementação, um sequencial(forma mais comum, está presente também na heuristica e na busca global), um paralelizado com openmp e o outro utilizando gpu para paralelizar o gpu ao invez de cpu.

Neste relatório essas cinco implementções serão analisadas com diferentes entradas para avaliar as alterções em suas saídas. Essas entradas possuírão diferentes tamanhos, alterando significamente. 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 apartir do MMS (o valor da pessoa com o menor valor), ou seja quanto maior o MMS maior a qualidade da saída.

2 Métodos Utilizados

2.1 Máquina utilizada

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

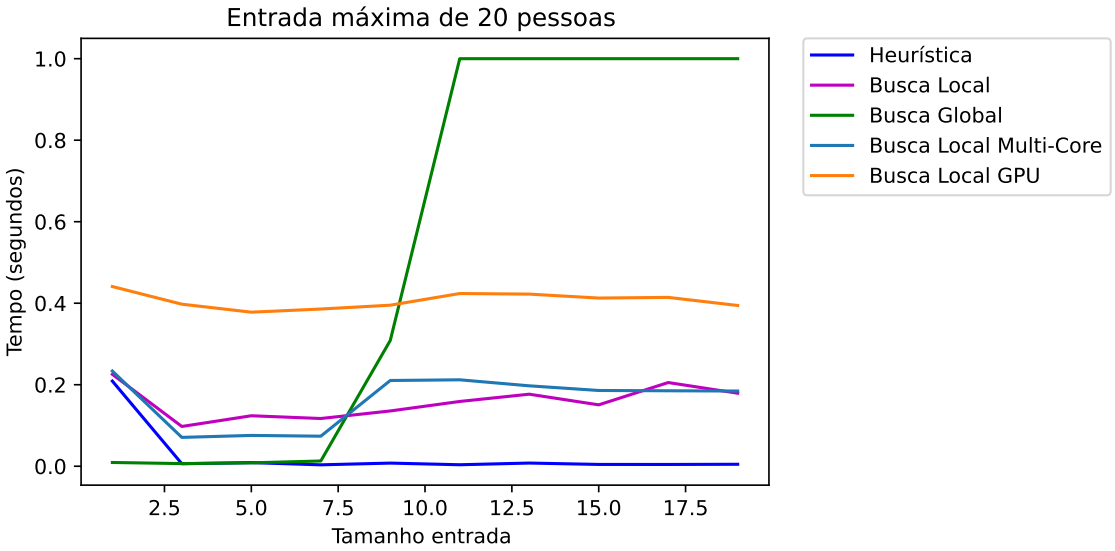
3 Efeito número de pessoas

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.

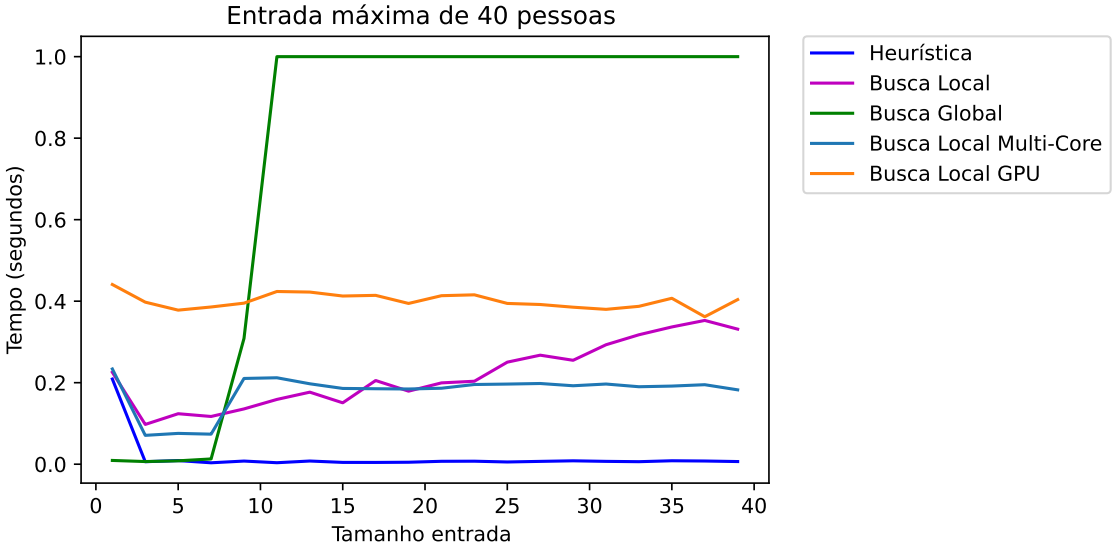
3.1 Tempo

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

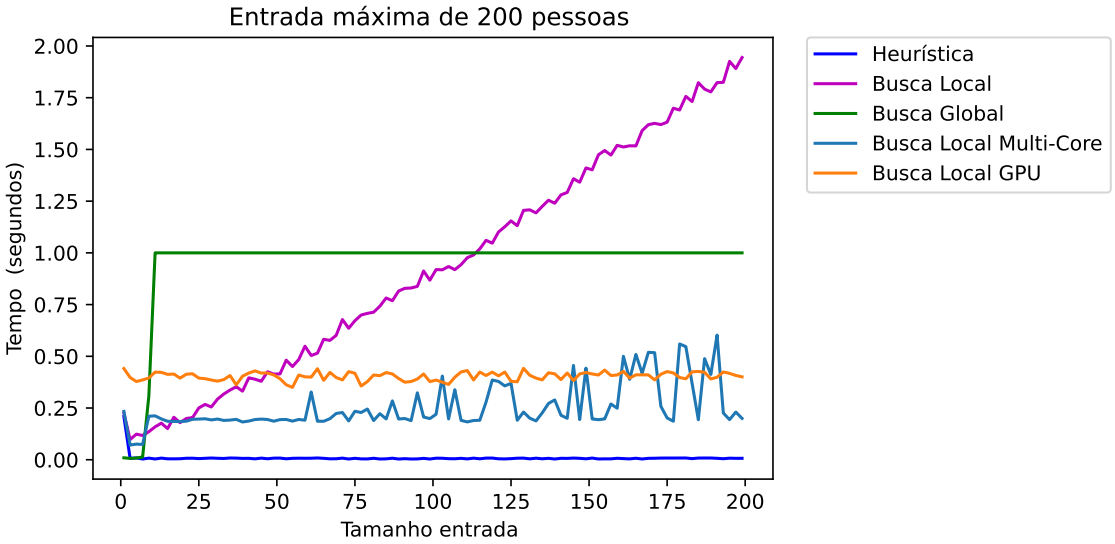
```
<matplotlib.legend.Legend at 0x7f4792a367c0>
```




```
<matplotlib.legend.Legend at 0x7f47929af2b0>
```




```
<matplotlib.legend.Legend at 0x7f47928de4f0>
```

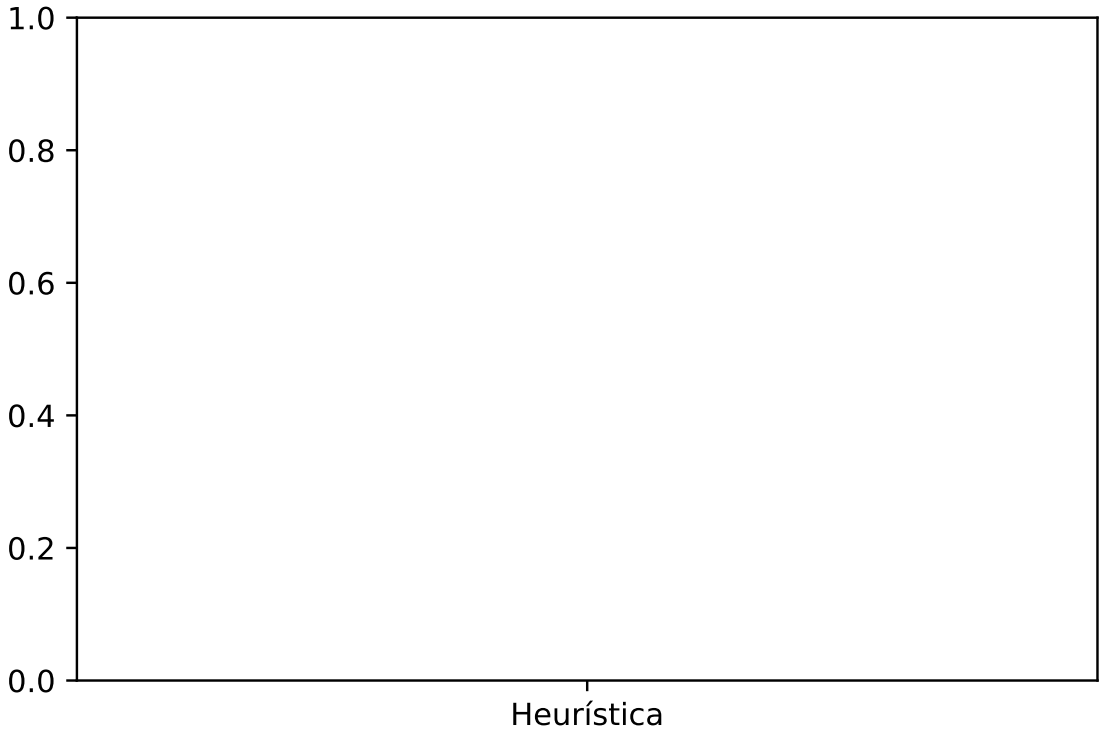


A partir dos gráficos pode-se concluir que os algoritmos de busca local de busca global possuem valores de tempo maiores do que a heurística, a qual não sofre fortes alterações. Um fato curioso que pode ser observado é que a busca local inicia com um tempo maior e logo antes da entrada de 10 - por volta 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 heuristíca é o de ordenar, por isso o tempo não sofre grandes alterações.

3.2 Qualidade da solução

```
---------------------------------------------------------------------------ValueE
Traceback (most recent call last)<ipython-input-1-9ff3f0c67f61> in
<module>
      6
      7 xbar = ["Heurística",'Busca Local','Busca Global','Busca Local
Multi-Core','Busca Local GPU']
----> 8 plt.bar(xbar,outsh,color ='maroon',width = 0.4)
      9
     10
~/.local/lib/python3.8/site-packages/matplotlib/pyplot.py in bar(x,
height, width, bottom, align, data, **kwargs)
   2649         x, height, width=0.8, bottom=None, *, align='center',
   2650         data=None, **kwargs):
-> 2651     return gca().bar(
   2652         x, height, width=width, bottom=bottom, align=align,
   2653         **({"data": data} if data is not None else {}),
**kwargs)
~/.local/lib/python3.8/site-packages/matplotlib/__init__.py in
inner(ax, data, *args, **kwargs)
   1359     def inner(ax, *args, data=None, **kwargs):
   1360         if data is None:
-> 1361             return func(ax, *map(sanitize_sequence, args),
**kwargs)
```

```
   1362
   1363         bound = new_sig.bind(ax, *args, **kwargs)
~/.local/lib/python3.8/site-packages/matplotlib/axes/_axes.py in
bar(self, x, height, width, bottom, align, **kwargs)
   2302                 yerr = self._convert_dx(yerr, y0, y,
self.convert_yunits)
   2303
-> 2304         x, height, width, y, linewidth, hatch =
np.broadcast_arrays(
   2305             # Make args iterable too.
   2306             np.atleast_1d(x), height, width, y, linewidth,
hatch)
<__array_function__ internals> in broadcast_arrays(*args, **kwargs)
~/.local/lib/python3.8/site-packages/numpy/lib/stride_tricks.py in
broadcast_arrays(subok, *args)
    536     args = [np.array(_m, copy=False, subok=subok) for _m in
args]
    537
--> 538     shape = _broadcast_shape(*args)
    539
    540     if all(array.shape == shape for array in args):
~/.local/lib/python3.8/site-packages/numpy/lib/stride_tricks.py in
_broadcast_shape(*args)
    418     # use the old-iterator because np.nditer does not handle
size 0 arrays
    419     # consistently
--> 420     b = np.broadcast(*args[:32])
    421     # unfortunately, it cannot handle 32 or more arguments
directly
    422     for pos in range(32, len(args), 31):
ValueError: shape mismatch: objects cannot be broadcast to a single
shape
```



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 eficiciente, 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.

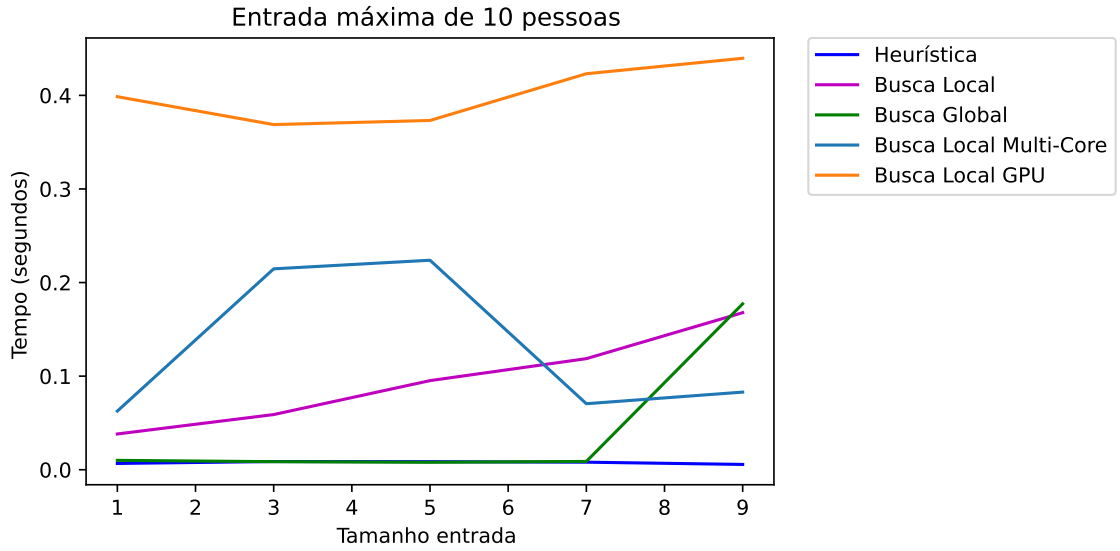
4 Efeito número de objetos

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.

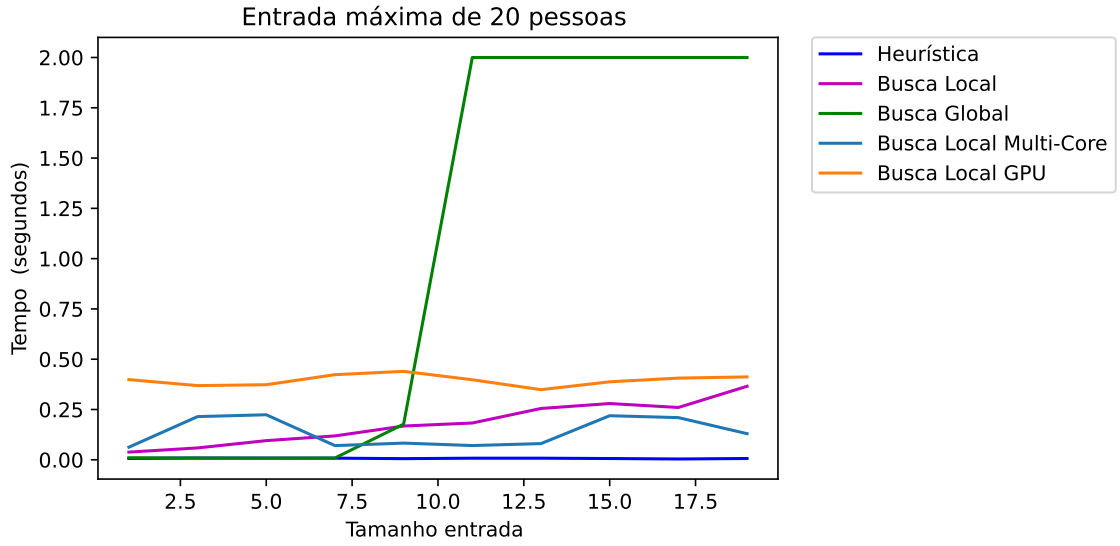
4.1 Tempo

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

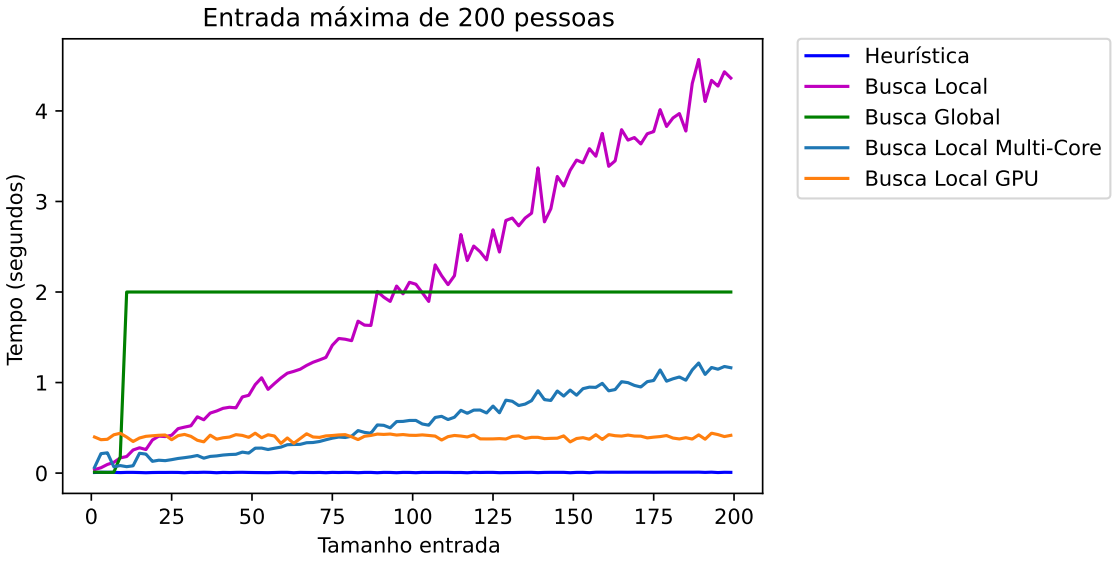
```
<matplotlib.legend.Legend at 0x7f4792130fd0>
```




```
<matplotlib.legend.Legend at 0x7f4792056fa0>
```




```
<matplotlib.legend.Legend at 0x7f4791f82790>
```



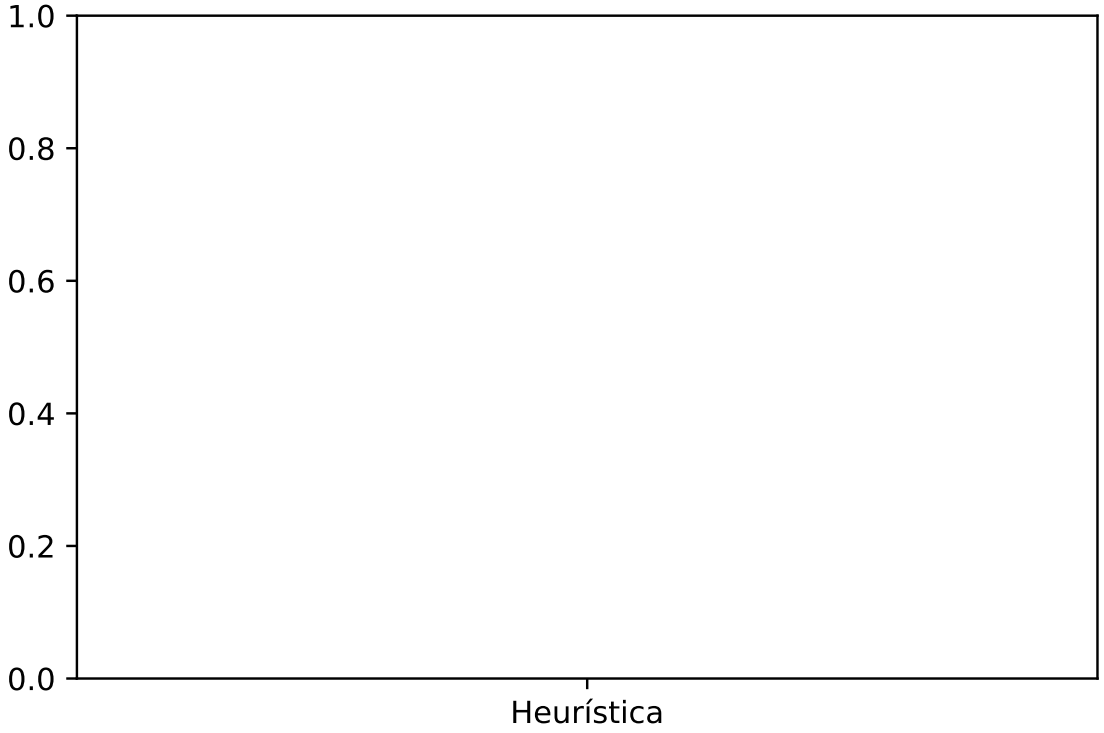
Como analisado no gráfico do tempo da seção anterior, a busca local e a busca global possuem uma variação significamente maior do que a heurística. Nas entradas pequenas as 3 implementações possuem tempos parecidos, mas há uma relação direta com o aumento da quantidade de objetos na entrada e com a aumento 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.

4.2 Qualidade da solução

4.2.1 Gráficos

```
---------------------------------------------------------------------------ValueE
Traceback (most recent call last)<ipython-input-1-7914c017d293> in
<module>
      5
      6 xbar = ["Heurística",'Busca Local','Busca Global','Busca Local
Multi-Core','Busca Local GPU']
----> 7 plt.bar(xbar,outsh,color ='maroon',width = 0.4)
      8
      9
~/.local/lib/python3.8/site-packages/matplotlib/pyplot.py in bar(x,
height, width, bottom, align, data, **kwargs)
   2649         x, height, width=0.8, bottom=None, *, align='center',
   2650         data=None, **kwargs):
-> 2651     return gca().bar(
   2652         x, height, width=width, bottom=bottom, align=align,
   2653         **({"data": data} if data is not None else {}),
**kwargs)
~/.local/lib/python3.8/site-packages/matplotlib/__init__.py in
inner(ax, data, *args, **kwargs)
   1359     def inner(ax, *args, data=None, **kwargs):
   1360         if data is None:
-> 1361             return func(ax, *map(sanitize_sequence, args),
**kwargs)
   1362
```

```
   1363         bound = new_sig.bind(ax, *args, **kwargs)
~/.local/lib/python3.8/site-packages/matplotlib/axes/_axes.py in
bar(self, x, height, width, bottom, align, **kwargs)
   2302                 yerr = self._convert_dx(yerr, y0, y,
self.convert_yunits)
   2303
-> 2304         x, height, width, y, linewidth, hatch =
np.broadcast_arrays(
   2305             # Make args iterable too.
   2306             np.atleast_1d(x), height, width, y, linewidth,
hatch)
<__array_function__ internals> in broadcast_arrays(*args, **kwargs)
~/.local/lib/python3.8/site-packages/numpy/lib/stride_tricks.py in
broadcast_arrays(subok, *args)
    536     args = [np.array(_m, copy=False, subok=subok) for _m in
args]
    537
--> 538     shape = _broadcast_shape(*args)
    539
    540     if all(array.shape == shape for array in args):
~/.local/lib/python3.8/site-packages/numpy/lib/stride_tricks.py in
_broadcast_shape(*args)
    418     # use the old-iterator because np.nditer does not handle
size 0 arrays
    419     # consistently
--> 420     b = np.broadcast(*args[:32])
    421     # unfortunately, it cannot handle 32 or more arguments
directly
    422     for pos in range(32, len(args), 31):
ValueError: shape mismatch: objects cannot be broadcast to a single
shape
```




```
  File "<ipython-input-1-868c3b8bc8a4>", line 6
    plt.bar(xbar,outsh,color ='maroon',width = 0.4)
    ^
IndentationError: unexpected indent
```

No último gráfico de qualidade da solução pode ser observado que com entradas pequenas os três algoritmos possuem resultados parecido. Contanto, com o aumento do número de objetos a diferença entre a busca local heurística vai se tornando cada vez maior. No priemiro 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 implementacões, se possível avaliar, tivessem resultados mais parecidos do que os da busca local e da heurística.

5 Conclusão

As implentações analisadas no relatório possuem algumas caracteristicas diferentes. Para entradas pequenas os algoritmos apresentam soluções com qualidades muito parecidas, e o tempo da heurística já inicia um pouco menor. Dessa forma essa 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.