Relatório Final

Maria Eduarda Bicalho

19 de Junho 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.

Os algoritimos recebem como entrada o número de pessoas, o número de objetos e os valores dos objetos. Eles retornam o MMS e os ids dos objetos de cada pessoa. Na técnica da Heuristica objetos são ordenados a partir dos valores e atribuidos para as pessoas nessa ordem. Já na busca local, um algoritmo de aleatorização é utilizado para atibuir os objetos. Depois é fica-se atribuir um objeto para a pessoa que possui o mms analisado até que não seja mais possível aumentar o mms. Na busca global. uma função recursiva é utilizada para checar todas as combinações possíveis e descobrir qual retorna o maior mms.

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.

1.1 Máquina utilizada

Máquina virtual insper

CPU op-mode(s): 32-bit, 64-bit CPU(s): 4 Thread(s) por core: 1 Model name: Intel(R) Xeon(R) Gold 5215 CPU @ 2.50GHz Hypervisor vendor: VMware Memória: 8152144 kB

2 Efeito número de pessoas

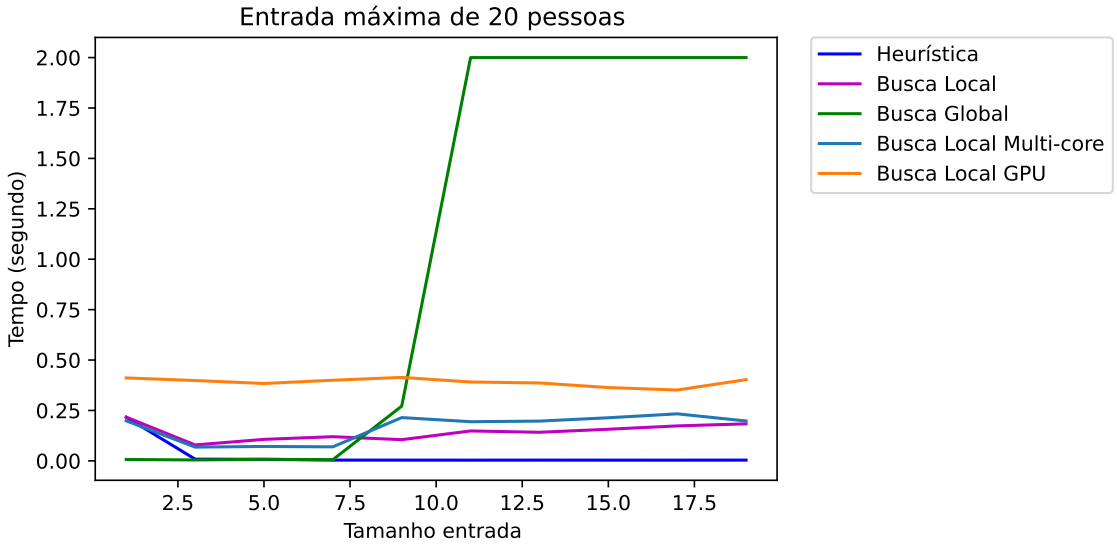
Nessa seção será realizada a análise do impacto de uma entrada com diferentes números de pessoas nos cinco diferentes algoritmos implementados no projeto. A capacidade da máquina foi sendo testada, aumentando cada vez mais o input de com o número pessoas. A variação do número de objetos foi baixa, para 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.

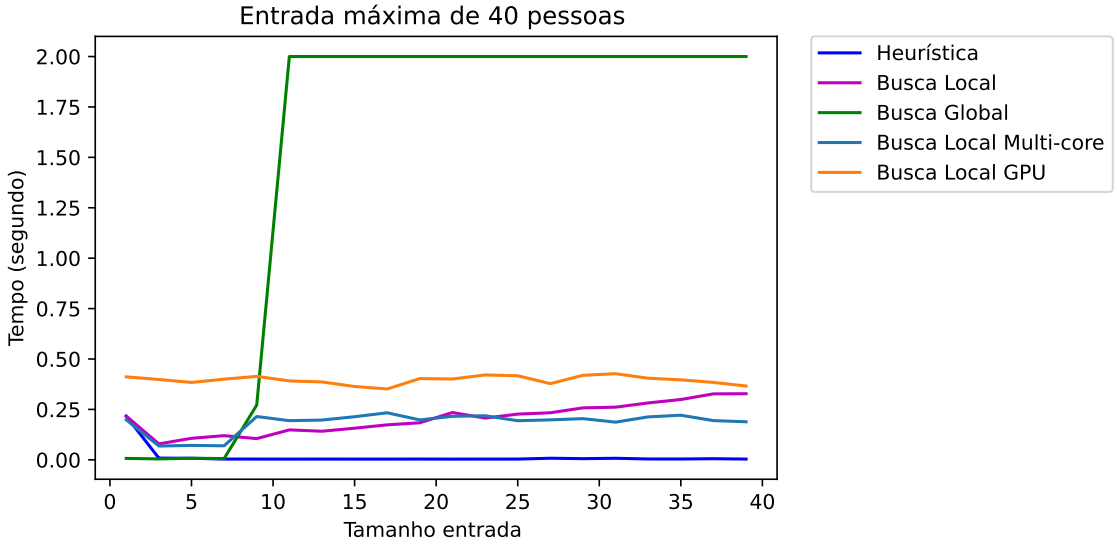
2.1 Tempo

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

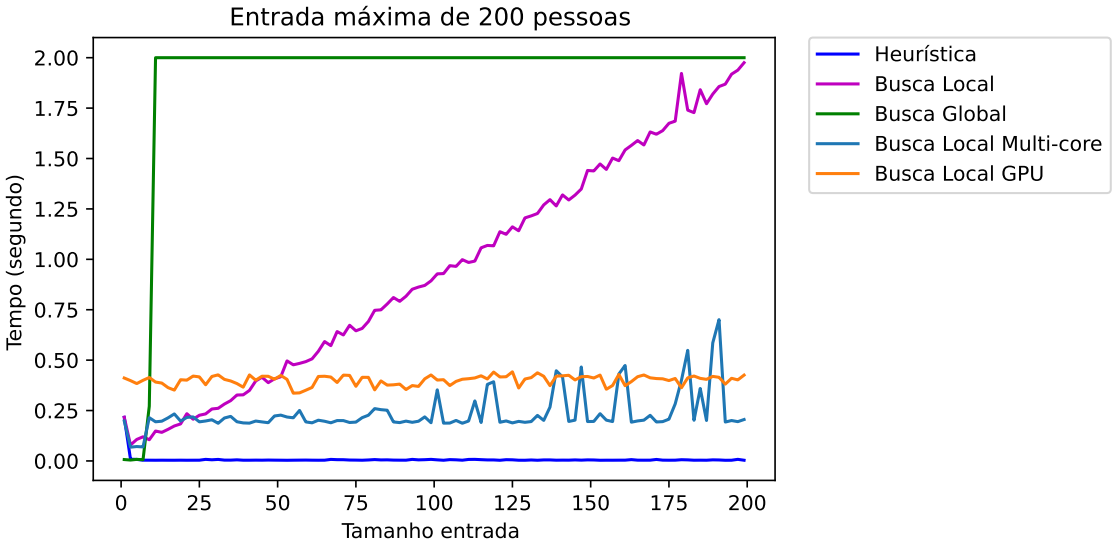
```
<matplotlib.legend.Legend at 0x7fa3794417f0>
```




```
<matplotlib.legend.Legend at 0x7fa379353160>
```




```
<matplotlib.legend.Legend at 0x7fa37927a790>
```

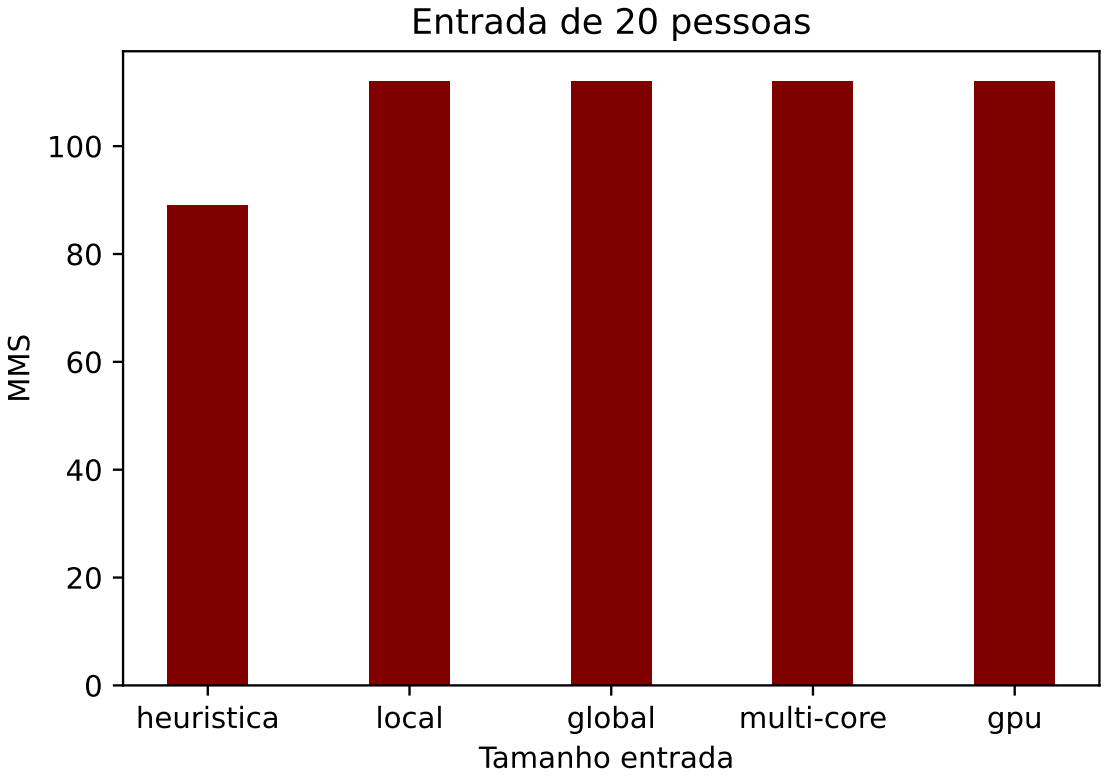


A partir dos gráficos pode-se concluir que os algoritmos de busca local de busca global possuem valores de tempo maiores do que a heurística, a qual não sofre fortes alterações. Um fato curioso que pode ser observado é que a busca local inicia com um tempo maior e logo antes da entrada de 10 - por volta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

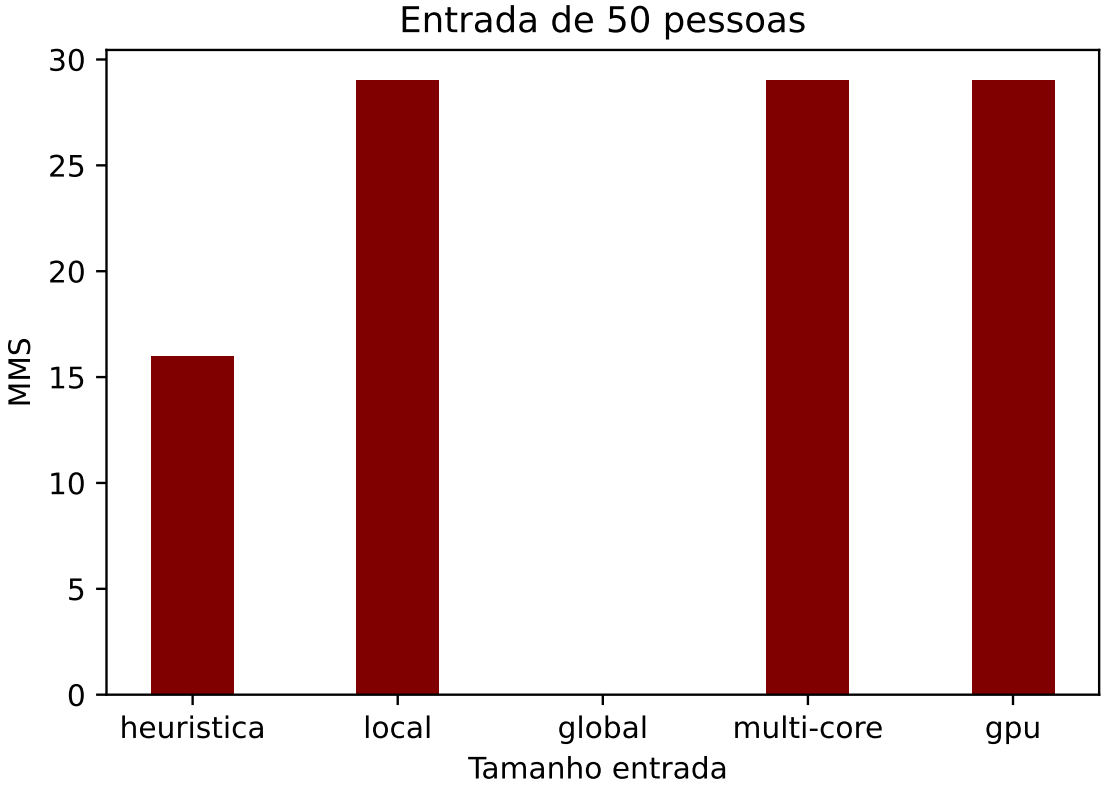
A partir do gráficos é possível notar que para entradas muito grandes, o código paralelizado com o openmp e como o programa que utiliza gpu tiverem um tempo de execução menor que a bsca local serial. É possível analisar também que o tempo do programa que utiliza a GPU se mantem quase contante em 0.5 segundos, estando maior que os outros nas entradas menores e menor que os outros nas entradas maiores - maior somente que a heurística. Esse resultado mostra que utilizar a gpu é boa para entradas maiores. O código paralelo do openmp mostrou uma diferença local inicial somente em entradas bem grandes como a de 100 pessoas. Acredito que isso ocorreu devido ao uso do critical, ferramente que aumenta bastante o tempo de execução quando utilizada.

2.2 Qualidade da solução

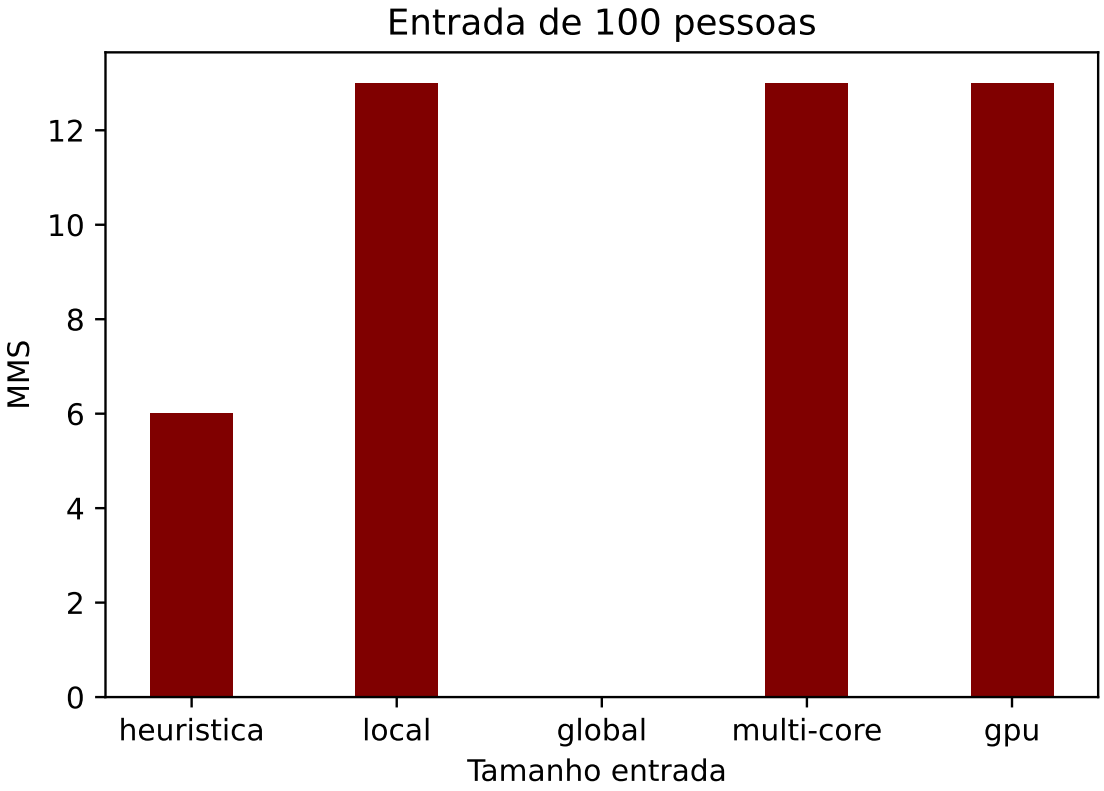
```
Text(0, 0.5, 'MMS')
```




```
Text(0, 0.5, 'MMS')
```




```
Text(0, 0.5, 'MMS')
```



A partir dos gráficos, é possível perceber que a algoritimo da heurística possui um resultado significamente menor que a dos outros. Com o aumento da entrada de pessoas, a diferença também aumenta. Como esperado, a implementação em GPU e a multi-core obtiveram resultados iguais da busca local serial. Como a o programa da busca global tem um tempo muito alto para entradas grandes, foi possível compará-la somente para a entrada de 10. Nesse gráfico, essa implementação teve resultado alto, igual da busca local.

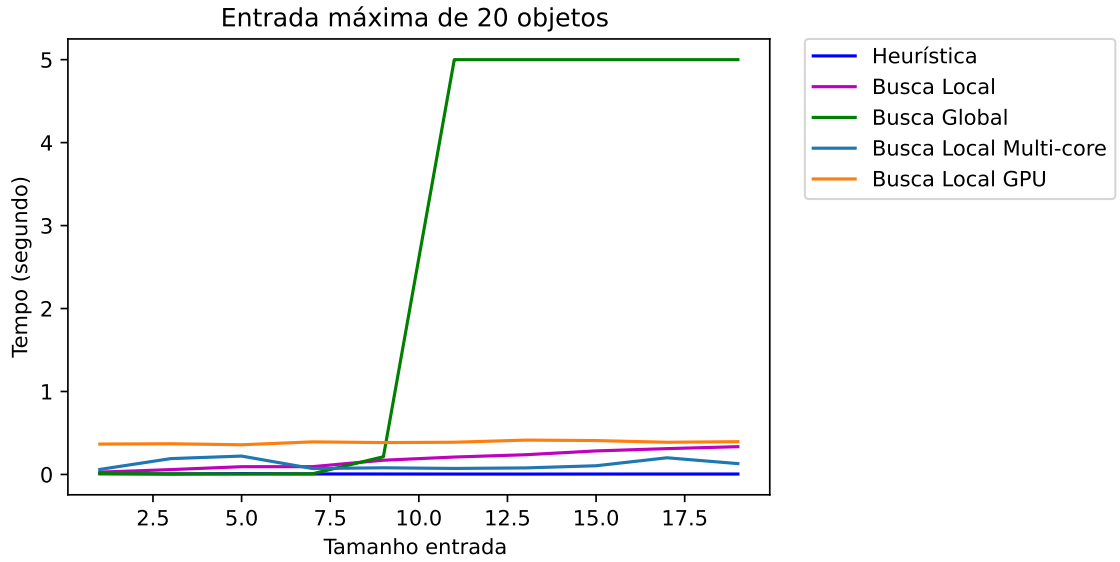
3 Efeito número de objetos

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.

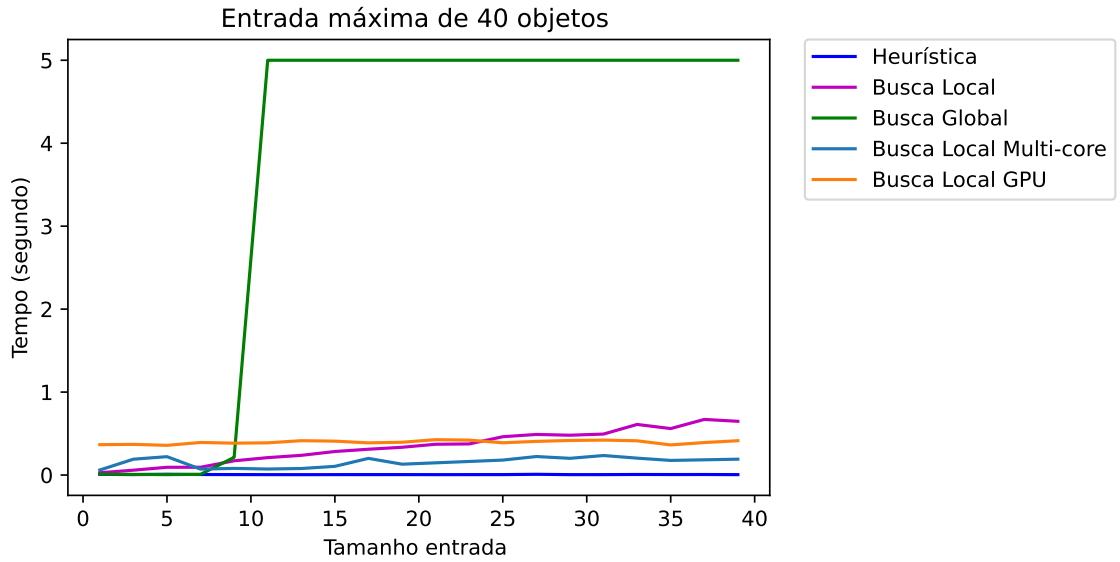
3.1 Tempo

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

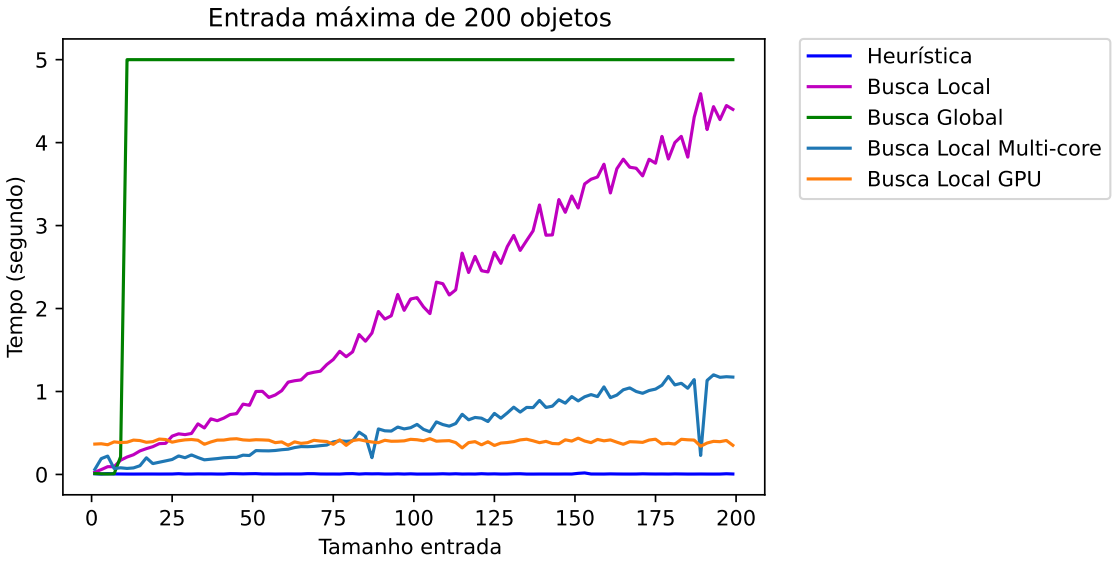
```
<matplotlib.legend.Legend at 0x7fa3790e1430>
```




```
<matplotlib.legend.Legend at 0x7fa378abcd00>
```




```
<matplotlib.legend.Legend at 0x7fa378a62e50>
```



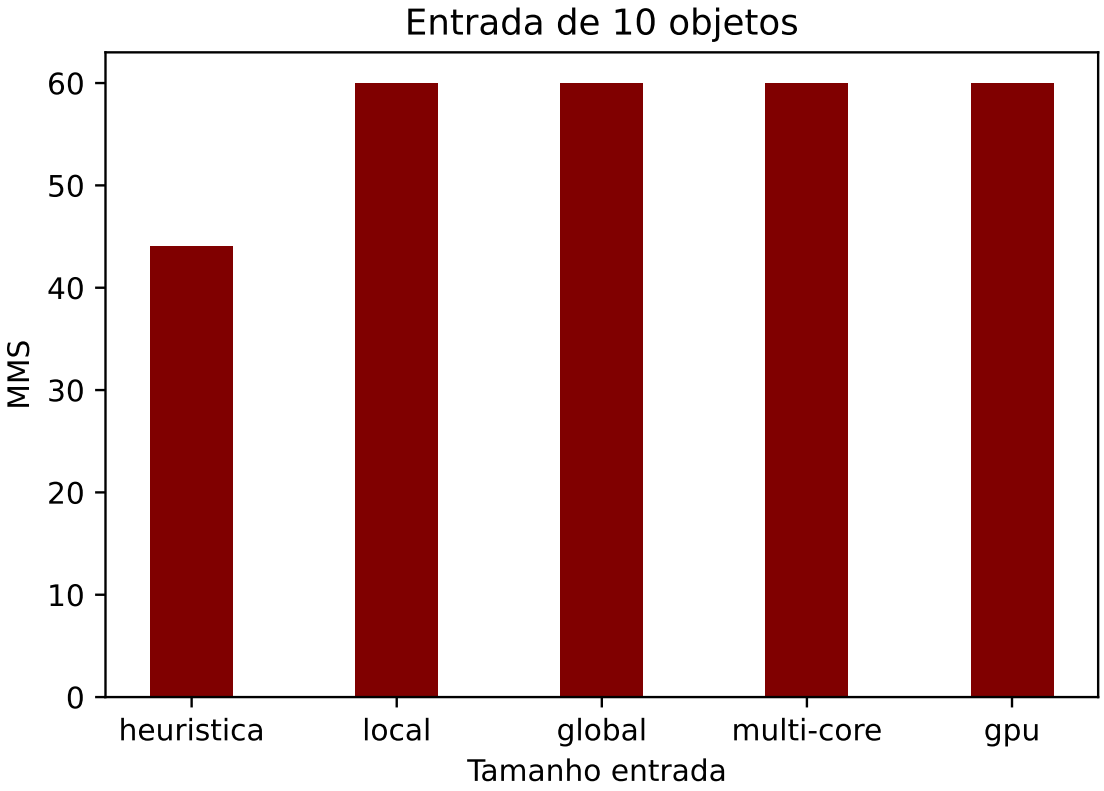
Como analisado no gráfico do tempo da seção anterior, a busca local e a busca global possuem uma variação significamente maior do que a heurística. Nas entradas pequenas as 3 implementações possuem tempos parecidos, mas há uma relação direta com o aumento da quantidade de objetos na entrada e com a aumento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

Como nas entradas com diferentes número de pessoas, o código paralelizado com o openmp e o programa que utiliza gpu tiverem um tempo de execução menor que a bsca local serial. É possível analisar também que o tempo do programa que utiliza a GPU novamente se mantem quase contante sempre menor que 1 segundo, estando maior que os outros nas entradas menores e menor que os outros nas entradas maiores - maior somente que a heurística. Esse resultado mostra que utilizar a gpu é interessante quando as entradas são maiores. O código paralelo do openmp mostrou uma diferença local inicial somente em entradas bem grandes como a de 100 pessoas. Acredito que isso ocorreu devido ao uso de critical, ferramente que aumenta bastante o tempo de execução quando utilizada.

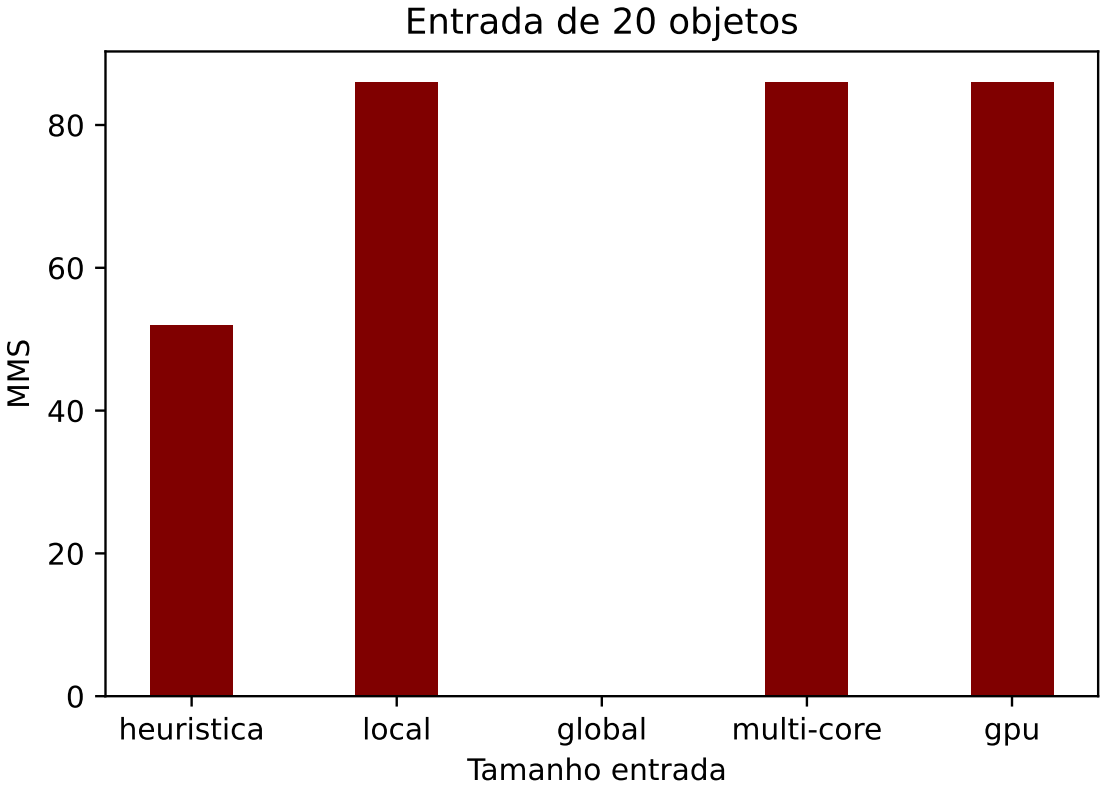
3.2 Qualidade da solução

3.2.1 Gráficos

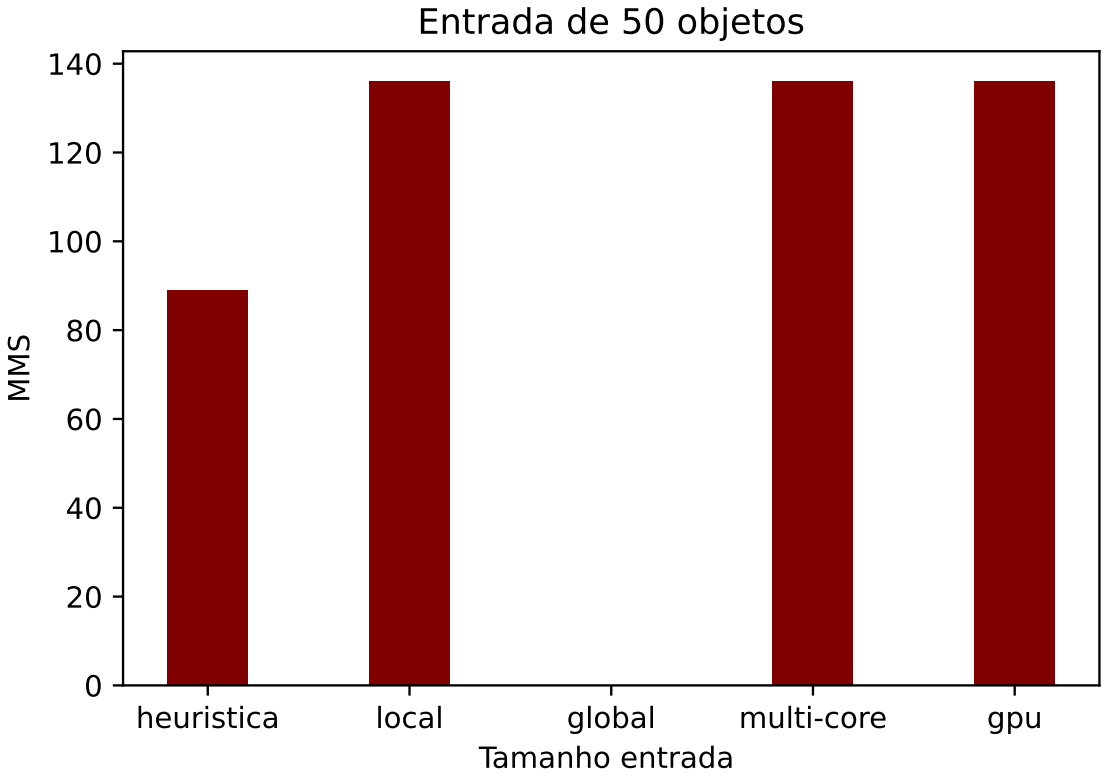
```
Text(0, 0.5, 'MMS')
```



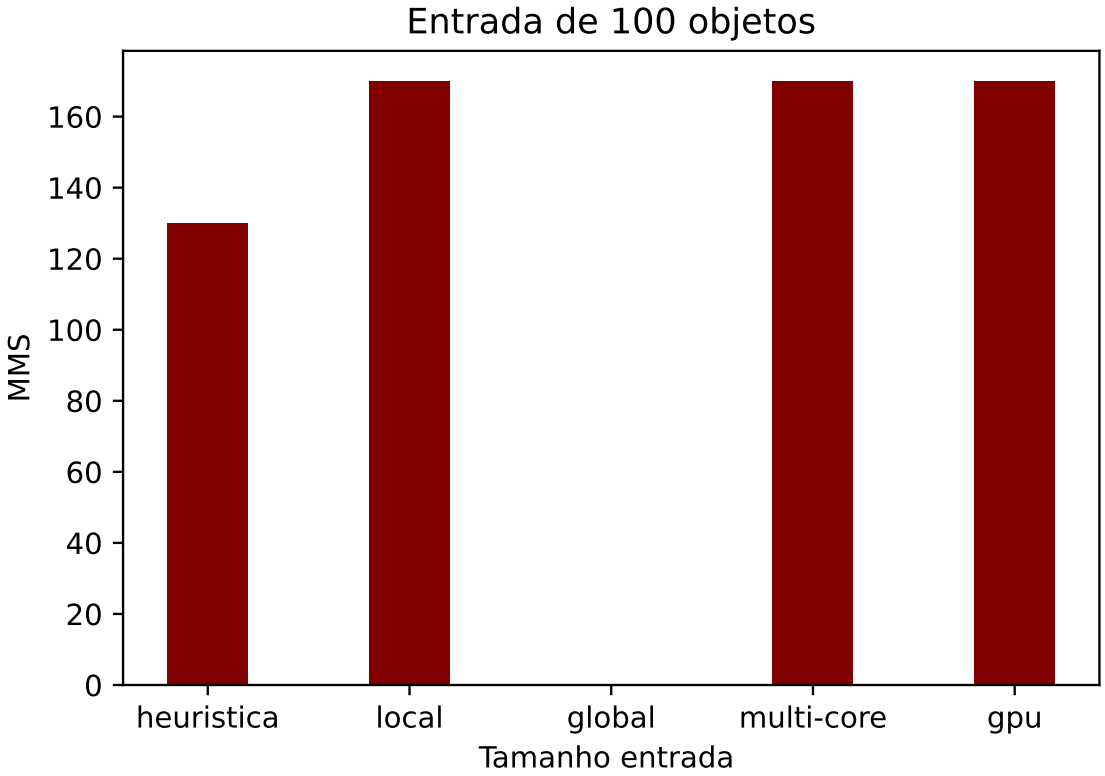

```
Text(0, 0.5, 'MMS')
```



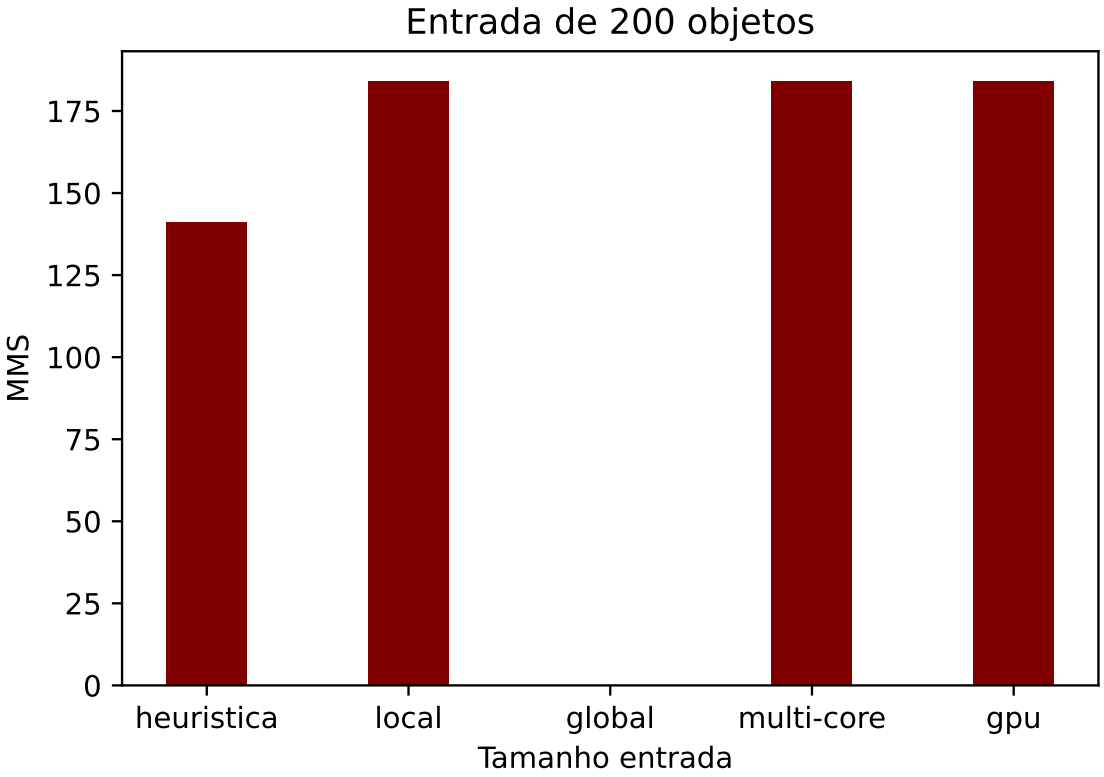

```
Text(0, 0.5, 'MMS')
```




```
Text(0, 0.5, 'MMS')
```




```
Text(0, 0.5, 'MMS')
```



A partir dos gráficos, mais uma vez, é possível observar que o algoritimo da heurística possui resultados piores que o dos demais. Nos gráficos de 10, 20 e 40 é possível perceber que com um aumento na entrada a diferença entre o resultado do programa da heurística e dos outros tende a aumentar. Isso deve acontecer devido a lógica para a Heuristica. Esse programa funciona melhor para entradas com valores de objetos parecidos. Com valores muito diferentes o resultado da heurística tende a piorar e com um número maior de objetos é mais provável que existam valores com diferenças maiores. A busca global, novamente, gera resultados somente para entradas menores. A solução foi a mesma da busca local, logo pode-se construir a hipótese que para entradas maiores geraria o mesmo resultado também. Como esperado, mais uma vez, os códigos paralelizados possuem mesmo resultado que a busca local.

4 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. Contudo, a busca global se mostrou a melhor escolha para entradas menores, uma vez que possui o maior resultado e menor tempo. Na medida que 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, principalmente as versões paralelizadas. A busca-local em gpu se mostru a mais adequada para entradas maiores, uma vez que essa solução não apresenta aumento no tempo com um aumento na entrada e sempre resulta na melhor solução. Consequentemente, as implementações podem ser utilizadas para diferentes finalidades, cada uma podendo ser escolhida a partir das preferências e importâncias de determinado projeto.