# Traffic Density Subtask 3

2019CS10343 and 2019CS10363

March 2021

## 1 Method1

In this method, we take x as the parameter, where x is the number of frames skipped. For example if x is 10, then we are processing every 10th frame of the video.

### 1.1 Metrics

For this method, the run-time metric is just the time taken to run the program. The baseline for this method is the code used in Assignment1 part2 i.e. processing every 5th frame. The utility metric is calculated by the following way:

The value of deviation is initialised at 0. First the output graph for the given value of x is found out. Then for every processed frame of the baseline, the absolute difference in the output of both dynamic and queue density is added to the value of deviation. After this deviation is divided by the total number of frames taken. We calculate utility as the inverse of this deviation.

### 1.2 Trade-off Analysis

For different values of x, the value of utility and run-time is calculated and then a graph is plotted with run-time on X-Axis and utility on Y-Axis.

From the graph it can be seen clearly that the value of utility increases as the run-time increases. The increase is almost linear. But for the cases when the run-time is low, i.e. when large number of frames are skipped, the graph is not the same. The reason for this may be that some frame will take more time to process than others. Hence it is possible that the case which has larger value of x may also have larger run-time but lower utility. However this is seen only for very large x such as 70 or 80 or greater than those.
For rest of the values, the graph is linearly increasing because the increase in runtime implies that there has been decrease in the value of x which means that more frames are being processed which leads to better accuracy and hence more utility. Hence the value of utility also increases with increase in runtime.

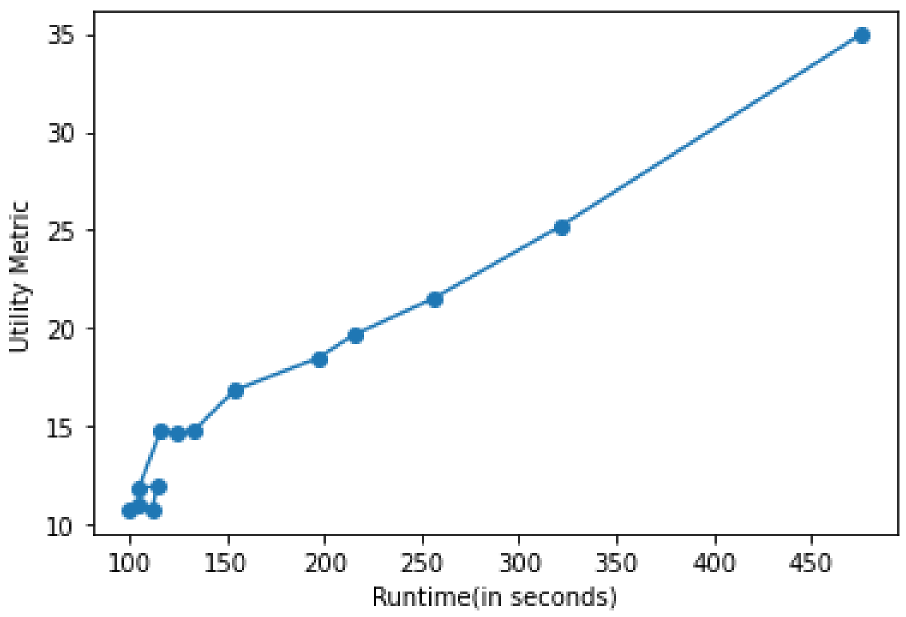

Figure 1: Utility vs Runtime for Method1

## 2 Method2

In this method, the parameters are X and Y, where X and Y are the values of resolution to which the frame is reduced and then processed.

### 2.1 Metrics

For this method, the runtime metric is just the time taken to run the program(in seconds). The baseline for this method are the resolution of the frames used in Part2 of the assignment.
The utility metric is found similarly as in Method1. The value of runtime and utility is noted for different values of X and Y parameters.

### 2.2 Trade-off Analysis

The graph is plotted with run-time on the X-Axis and utility on the Y-axis.

From the graph it can be clearly seen that the value of utility increases with increase in the runtime. This is because the runtime is higher for the frames having higher resolution. They may take more time in running but give more accurate results and hence more utility. Hence utility is more for the frames with higher resolution.
However, it can be seen from the graph that the increase is not linear. For some values, there is sudden increase in the utility whereas for the others, there is

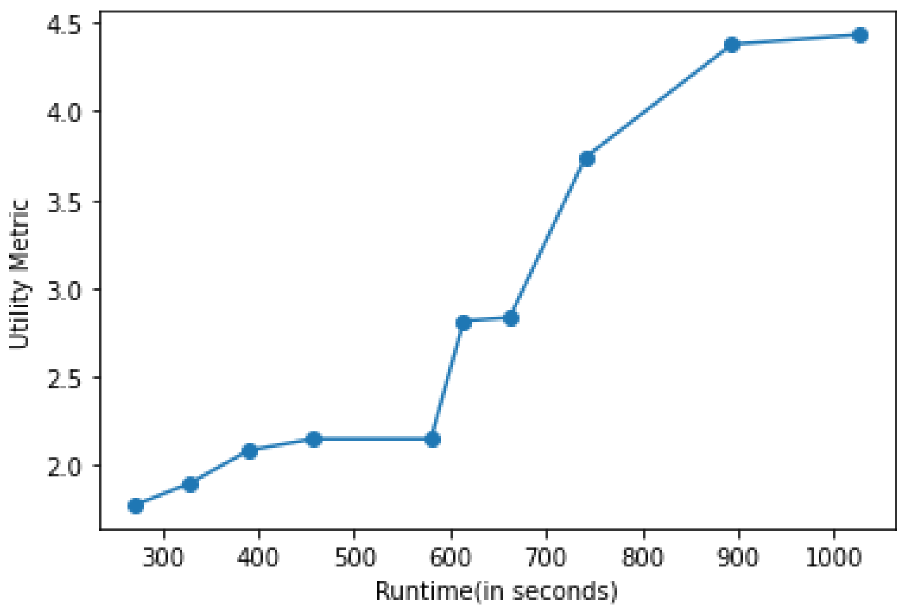

Figure 2: Utility vs Runtime for Method2

minimal change. The slope of the graph at various points varies. This may be because for a certain value of X and Y, the reduction in resolution may lead to an output which is very different from the baseline. In that case there is high slope. While there may be another case where the resolution produces an output similar to the next value of X,Y. In that case there is low value of slope.

## 3 Method 3

In this method, we will split work spatially across threads (application level pthreads) by giving each thread part of a frame to process. Parameter can be number of splits i.e. number of threads, if each thread gets one split. You might need to take care of boundary pixels that different threads process for utility. For example, if we give split as 4 then frames will get divided into 4 parts(for simplicity we have divided frames equally) and these parts will get processed in 4 different pthreads.

### 3.1 Metrics

For this method, the run-time metric is just the time taken to run the program. The baseline for this method is when the split given is 1 ( command line argument) and it will give same data set as subtask 2. The utility metric is calculated by the following way:

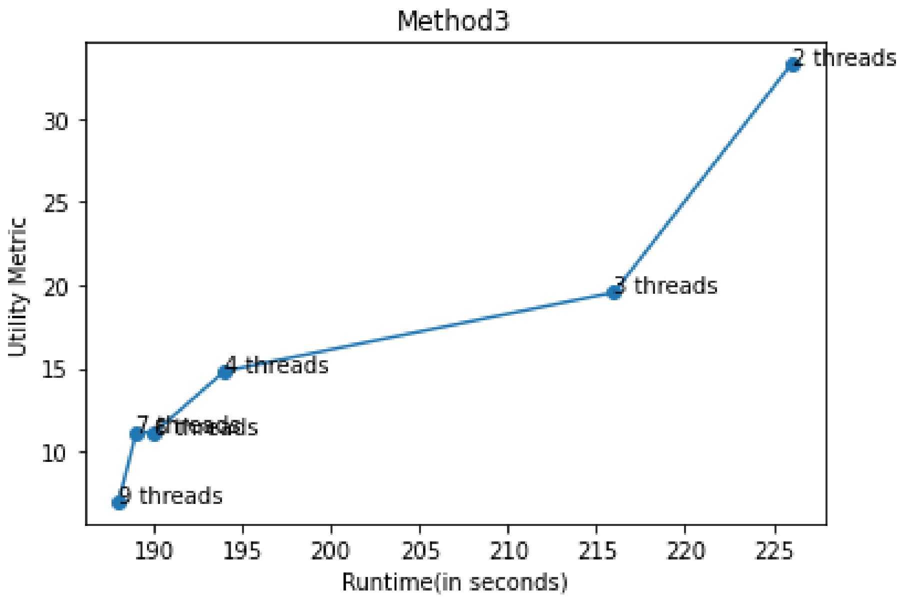

Figure 3: Method 3 : Utility v/s Run-time

The value of deviation is initialised at 0. First the output graph for the given split value (command line argument) is found out. Get the data set for different split values. Then for every processed frame of the baseline, the absolute difference in the output of both dynamic and queue density is added to the value of deviation. After this deviation is divided by the total number of frames taken. We calculate utility as the inverse of this deviation.

## 3.2 Trade-off Analysis

For different values of split (number of parts in which we will divide our frame) different data set values having different execution times is shown. For a given number of split, the parts of a frame processed on a particular pthread will have same execution time. The execution time is machine dependent. As we increase the number of splits and the environment of run time is kept constant then the execution time for densities are variable . This is because as the split increases more number of parts of same frame get processed in parallel pthreads. Also due to large splits the time for splitting the values cannot be ignored.
From the graph, we observe that as the number of splits increases the run time of code decreases resulting in decrease in utility metric. The parts of frame are run in parallel resulting in lower execution time. Although the number of splits increases some time but the overall execution time is less but smaller resolution frame takes less time for comparison. The execution time is machine dependent.

## 4 Method 4

In this method, we will split work temporally across threads (application level pthreads), by giving consecutive frames to different threads for processing. Parameter can be number of threads. For example, if number of pthreads are 4, then 4 consecutive frames will get into parallel processing.

### 4.1 Metrics

For this method, the run-time metric is just the time taken to run the program. The baseline for this method is when the number of pthread is 1 i.e. it will have similar data set as of subtask 2.The utility metric is calculated by the following way:
The value of deviation is initialised at 0. First the output graph for the given pthread value (command line argument) is found out. Get the data set for different values of pthread. Then for every processed frame of the baseline, the absolute difference in the output of both dynamic and queue density is added to the value of deviation. After this deviation is divided by the total number of frames taken. We calculate utility as the inverse of this deviation.

### 4.2 Trade-off Analysis

Different values of pthread will yeild different data set values having different execution times. For a given number of pthread, the frames processed on pthreads will have same execution time. The execution time is machine dependent. As we increase the number of pthreads and the environment of run time is kept constant then the execution time for queue density shows gradual decrease in value. This is because as the pthread increases more number of frames get processed in parallel.
From the graph, we observe that as the run time of code increase, utility metric also increases. Run time increases with decrease in number of pthreads. As the pthread increases , ideally the run time should decrease. Large number of pthreads will execute the frames in parallel processing and hence run time reduces. This run time is machine dependent.

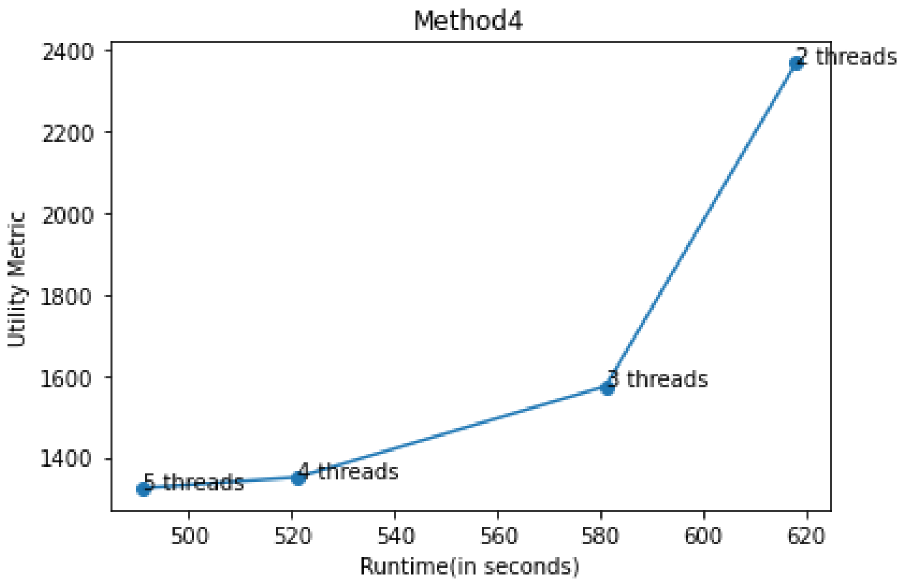

Figure 4: Method 4 : Utility v/s Run-time