A FAST WEIGHTED MEDIAN ALGORITHM BASED ON QUICKSELECT

André Rauh and Gonzalo R. Arce

University of Delaware
Department of Electrical and Computer Engineering
rauh@ee.udel.edu, arce@ee.udel.edu

ABSTRACT

Weighted median filters are increasingly being used in signal processing applications and thus fast implementations are of importance. This paper introduces a fast algorithm to compute the weighted median of N samples which has linear time and space complexity as opposed to $O(N \log N)$ which is the time complexity of traditional sorting algorithms. The proposed algorithm is based on Quickselect which is closely related to the well known Quicksort. We compare the runtime and the complexity to Floyd and Rivest's algorithm SELECT which to date has been the fastest median finding algorithm and show that our algorithm is up to 30% faster.

***Index Terms*—** Median filters, Algorithms

1. INTRODUCTION

Weighted median (WM) filters are used in a variety of signal processing tasks such as image denoising, compressive sensing [1] and interpolation. When impulsive noise is present WM Filters are often used since linear filters break down. Medians and order statistics –a special case of WM– are also used in the statistical analysis of large datasets.

The problem of estimating a constant parameter β under additive noise given N observations can be solved by minimizing a cost function under different error criteria. The sample median follows from minimizing the error under the L_1 error norm. Letting the input samples have different variances the weighted median is derived [2]. Given N samples and weights the question is which sample to choose in order to minimize the cost function? It can be shown that a simple procedure is to first sort the samples with their corresponding weights and then take the partial sum of the weights until this sum is larger than half the sum of all weights.

It is well known that sorting an array of N elements requires $O(NlogN)$ comparisons. However, using a similar idea as in Quicksort, Hoare [3] introduced Quickselect which is an average linear time algorithm. The runtime of both algorithms greatly depend on the choice of the pivot elements which are used for partitioning the array. This paper shows a new way to choose the pivots and thereby improve the performance of Quickselect. Moreover we extend the concept of Quickselect which seeks the kth order statistic (OS) to the more general case of weighted median filters.

This work was partially funded by the National Science Foundation (NSF) under grant EECS-0725422 and CIF-0915800.

This paper first reviews the definition of weighted medians briefly. The standard Quickselect algorithm is reviewed and it is shown how to extend it to solve the weighted median problem. The main idea of how to select the pivots is then explained in detail. Simulation results of a C–implementation are included in section 5. The appendix contains the pseudocode of the algorithm.

1.1. Weighted Median Review

Definition 1 *Let $\{X_i\}_{i=1}^N$ be a set of N samples and let $\{W_i\}_{i=1}^N$ be their associated weights. Now rearrange the samples in the form $X_{(1)} \leq \cdots \leq X_{(N)}$ then $X_{(k)}$ is referred to as the kth OS. Further denote $W_{[k]}$ as the associated weight of $X_{(k)}$.*

For sake of simplicity we assume throughout this paper that all weights are positive. All results can easily be extended to allow negative weights by coupling the sign of the weight to the corresponding sample and use the absolute value of the weight [2]. It can be shown, e.g. in [2], that given a set of N samples and N weights the output of the weighted median is determined as follows:

$$\hat{\beta} = \left\{ X_{(k)} : \min k \quad \text{for which} \sum_{i=0}^{k} W_{[N-i]} \geq W_0 \right\}$$

where $W_0 = \frac{1}{2}\sum_{i=1}^{N} W_i$. Note that finding the kth OS is a special case of the above definition and can be found by replacing W_0 by $N-k$ and set all weights to 1.

2. THE QUICKSELECT ALGORITHM

Quickselect was first introduced in [3] as an algorithm to find the kth OS. Note that Quicksort and Quickselect are similar and the only difference between the two is that the former recurs on both subproblems –the two sets after partitioning– and the latter only on one.

Quickselect finds the kth OS of the samples in the array of N samples X by first choosing a sample at random from X

which is called the pivot. By comparing all other elements to the pivot the rank r of the pivot is determined. The pivot is then put into the rth position of the array and all other elements smaller or equal than the pivot are put into positions before the pivot and all larger elements are put after the pivot. This step is called partitioning and can be implemented very efficiently by running two pointers towards each other. One from the beginning of the array and one from the end, swapping elements if necessary until both pointers cross each other. If $r > k$ then the kth OS is contained in the first part of the array and Quickselect recurses on this part. If $r < k$ then Quickselect recurses on the second part but instead searches for the $(k-r)^{\text{th}}$ OS. If $k = r$ the recursion terminates and the pivot is returned.

For odd N the special case of $k = (N+1)/2$ is the well–known median and can also be considered as a special case of the WM with all weights equal to one. A minor modification of Quickselect can be done in order to find the WM in the general case with arbitrary weights. Instead of counting the number of elements less than or equal to the pivot (which is the same as the rank) the algorithm sums up the weights of all the samples which are less than or equal to the pivot. This sum of weights is then compared to W_0 and before the recursion is done all the weights of the discarded samples are assigned to the pivot which now functions as a proxy for all the discarded samples.

3. NEW PIVOT SELECTION STRATEGIES

The runtime of the algorithm depends greatly on the choice of the pivot. Consider an example: If the pivot is small compared to the sought WM then only elements which are less than the pivot are discarded. In the worst case –i.e. if the pivot is the smallest element of the set– no elements are discarded. However, the main cost of the partitioning step is to compare all $N-1$ elements to the pivot. Clearly, a pivot close to the actual WM is desired. Assuming no prior knowledge of the sample distribution or their weights, the only good "guess" for a pivot would be to choose the median of the samples. This ensures that exactly $(N-1)/2$ samples are removed. However finding the median is itself a selection type problem. The idea is to use an approximation of the median as the pivot. A straightforward approach is to take a random subset of X and find the median of this smaller set. How large this subset should be in the optimal case was studied in [4]. The proposed algorithm however will not use the results in [4] but instead will use data fitting to find out the subset size which minimizes the runtime for each N. After extensive simulation, data was gathered and Eureqa [5] was used to fit the data.

For a large number of samples it is unlikely to find the exact WM as the first pivot, thus it is assumed that the first pivot was either larger than or smaller than the sought WM. Without loss of generality we assume throughout this paper that the first pivot was smaller than the WM, the other case follows from symmetry. The next question is how to choose the second pivot. Should the median of a subset be used again? Intuitively, if the first pivot was smaller but close to the WM then a good choice for the second pivot is an element close to the WM but slightly larger than it. If we select the median of a subset again we will most likely be far away from the WM, thus not discarding many elements. It is natural to use the approach of using the kth OS of a subset as the second pivot. To find the optimal k we minimize the expectation of an approximate cost function:

$$E\{T_k\} = k\frac{N_1}{M_1}(1-P_k) + (N_1 - (k-1)\frac{N_1}{M_1})P_k$$

where N_1 is the number of samples left after the first iteration and M_1 is the cardinality of the random subset of the remaining samples. P_k is the probability that the kth OS of the random subset is less than $\alpha = (W_0 - W_{\leq}^1)/(2W_0 - W_{\leq}^1)$, where $X_{\leq}^1 = \{X_i \in X | X_i \leq p_1\}$ and p_1 is the first pivot. Note the cost was defined as the number of samples left after the next partitioning step is done. By multiplying with factor M_1/N_1 we can equivalently minimize

$$E\{T_k\} = k + (M_1 - 2k + 1)P_k$$

The cost function cannot be solved analytically and therefore the algorithm uses the central limit theorem to approximate the discrete derivative of P_k in combination with Newton's method to find the k which minimizes the cost function.

Finding a good starting point for Newton's method is crucial for this function since otherwise the method diverges. Therefore simulations were performed to find the optimal k for different M_1 and α. The symbolic fitting program Eureqa [5] was used to find a function which was then used to determine a good starting point. Since this starting point is already very close to the actual minimum, Newton's method only needs very few (1-2) iterations to converge to the optimal point. For extremely small values of α the poisson approximation was used instead of the normal approximation. Given that the optimal k is now known, Quickselect is called recursively to get the pivot for this second iteration. With high probability this pivot will be slightly larger than the WM and hence a lot of samples will be discarded. Though, it can happen that the pivot was smaller than the WM and then –the bad case– only very few samples are discarded. Note that in the good case we have a bounded problem, i.e. we know that the WM is between the first and the second pivot. Therefore in the bad case we repeat the second iteration step until the problem is bounded. This is necessary for the pivot computation strategy which will be used after the problem is bounded.

From this point on, the algorithm computes pivots using the strategy

$$p = cX_{\min} + (1-c)X_{\max}$$

where p is the pivot, c is some constant between 0 and 1 and $X_{\min}$ and $X_{\max}$ is the minimum and maximum points left in the array. Note that this strategy is very different to the previous ones since the pivots are not selected anymore but computed which implies that the pivot is most likely not an element of the set. Note however, that it will still be possible to partition the set. Just like before the pivot will be inserted into the set and take over the weights of the discarded samples. Note that this will introduce new samples into the set which originally did not exist. However this will not change the WM and therefore not change the result.

An optimal c at some iteration can again be found by minimizing a cost function which will be very similar to the above cost function. In practice however it turns out that the optimal solution performs worse than a suboptimal crude approximation. This can be easily explained since the overhead involved in using Newton's method is not worth the gain compared to a suboptimal choice. The following formulas are therefore used:

$$\begin{aligned}\beta &= \frac{W_0 - W_{\min}}{2W_0 - W_{\min} - W_{\max}} \\ f(x) &= x - x^2 + x^3 + 1.4x^4 \\ c &= 0.5 - f(0.5 - \beta)\end{aligned}$$

This formula holds for $\beta \leq 0.5$ but a similar formula can be stated for the symmetric case $\beta > 0.5$. Intuitively, if there are many samples between $X_{\min}$ and $X_{\max}$ then this modified interpolation search will work very well. If the number of samples in the data set decreases however, it is then more likely to pick the wrong pivot and the performance degrades. Therefore the algorithm stops when the problem size gets below a certain threshold level and solves the remaining problem by the standard median–of–3 Quickselect implementation. Also, in the case that this approach does not remove any elements –this can happen for some inputs– then the algorithm falls back to the median–of–3 Quickselect as well to protect from an infinite loop.

4. COMPLEXITY ANALYSIS

The space complexity of the algorithm is $O(N)$ since the partitioning step can be done in–place and hence does not need additional memory. For the first and second partition step the algorithm needs to choose the pivot from a random subset of X. However, this approach would require additional memory and time to generate the random numbers. In practice it is more efficient to sample uniformly. Selecting an OS from this uniformly subset can also be done in–place and hence there is no need for extra memory.

Since the most time is spent in the partitioning step the time complexity is defined as the number of element–wise comparisons the algorithm makes until termination. In addition the comparisons to select the pivots recursively have to be taken into account. Using a median–of–M samples as the pivot, it was shown in [4] that the number of comparisons is $2N + o(N)$ on average. Simulations in the next section show that the proposed algorithm beats this result. This is mainly due to the strategy used for selecting the second pivot. As explained above selecting just the median is not optimal in general.

Table 1. Number of comparisons are not affected by heavy tailed input distributions.

α	2	1.8	1.4	1.0	0.8	0.6	0.4
μ	1.74	1.74	1.74	1.74	1.74	1.74	1.74

The worst–case runtime of the proposed algorithm is $O(N^2)$ trivially since the algorithm will always do better or equal than the median–of–3 Quickselect which has the same worst–case runtime. In [4] is shown that the worst–case runtime for a median–of–M approach is $O(N^{3/2})$ for $M = \theta(\sqrt{N})$ which is the same approach used for the proposed algorithm. This is an indication that the proposed algorithm's worst–case complexity is also better than N^2. However analytic worst–case analysis has yet to be done for the algorithm.

5. SIMULATION RESULTS

To compare the simulation results among different sample set sizes the number $\mu = C/N$ is used. C is the number of comparisons and N the input data set size. The proposed algorithm was implemented in MATLAB and simulated with different input distributions of X. Since weighted medians are often used with heavy tailed input distributions the algorithm was run on α–stable distributions. The results are depicted in Table 1. Note that the mean does not change and is robust to heavy tailed inputs. This can be easily explained: After the first two steps the algorithm makes the problem bounded above and below and thus all outliers are already discarded. Furthermore due to the approach to pick the first two pivots it is highly unlikely to select an outlier as the pivot.

We chose to compare the performance to two other algorithms. The standard median–of–3 Quickselect and Floyd and Rivest's SELECT algorithm [6]. Since both algorithms only solve the more specific problem of finding the OS of a set of samples a version of the proposed algorithm which finds the median was implemented in C. The results in terms of number of comparisons are depicted in Fig. 1. The proposed method clearly outperforms SELECT for smaller and medium input sizes and both methods converge to the $1.5N$ optimum for very large input sizes. The $1.5N$ optimum can be explained easily: For very large input sizes the number of samples removed is almost $N/2$ after the first partitioning step which does N comparisons. The second pivot will also remove almost $N/2$ samples which cost $N/2$ comparison to partition. All what is left is a fraction of the input size and therefore the number of comparisons converge to $1.5N$. To explain why the proposed algorithm performs better we have to look at how SELECT works: SELECT first chooses two pivots which lie with high probability just to the left and just to the right of the sought median. This however is suboptimal unless the input size is very large since the two pivots have

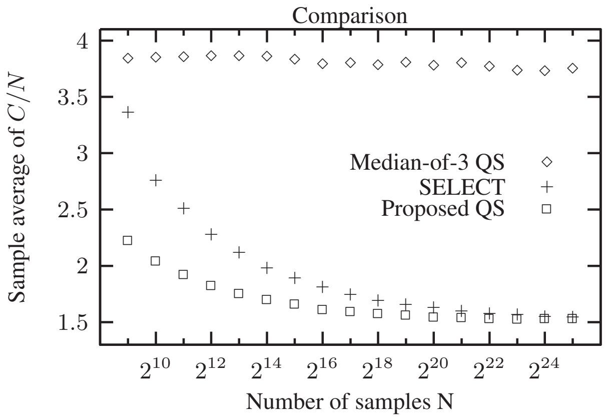

Fig. 1. Sample average of C/N for the different algorithms.

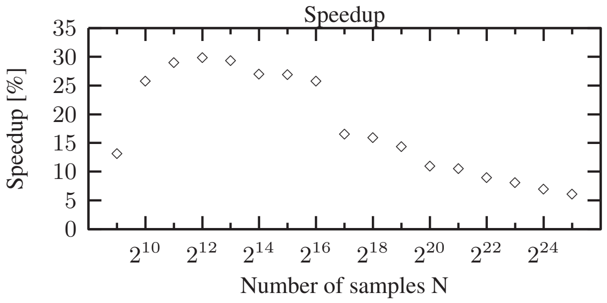

Fig. 2. Speedup against Floyd and Rivest's SELECT.

to be chosen to be relatively far away from the median and therefore many unnecessary comparisons are done. The proposed method on the hand tries to get the first pivot as close as possible to the median and then chooses the second pivot such that the median is with high probability between the first and the second pivot. This explains Fig. 1.

However since the proposed algorithm requires more code than the other two algorithms it is of interest to show how the runtime of the proposed algorithm compares. For these simulations, C–implementations where run and timed and the sample averages are compared. Since SELECT was always faster than the standard Quickselect for the used input sizes we compared the speedup against SELECT which is depicted in Fig. 2. The proposed method is up to 30% faster than SELECT for a wide range of input sizes and converges to 0 for very large input sizes which is expected since both are optimal for large N. The speedup for the input sizes 513, 1025 is smaller due to the involved overhead of the more complex implementation. This increased complexity however pays off for larger input sizes. For this reason the proposed method is slower than SELECT for input sizes smaller than 513 samples. For all simulations the input samples were i.i.d. normal distributed.

6. CONCLUSIONS

A fast new WM algorithm is introduced which is based on Quickselect. The new idea of selecting the first and second pivots as well as computing the pivots after the problem is bounded was presented. Theory explains that the proposed method should be faster than Floyd and Rivest's SELECT unless input sizes are very small. This was backed up by experiments which showed a speedup of up to 30%. A C–implementation can be downloaded from the homepage of one of the authors: *http://www.eecis.udel.edu/~rauh/fqs/*

A. ALGORITHM PSEUDOCODE

We first state the standard Quickselect algorithm in pseudocode followed by a table which describes how to choose the pivots.

```
name  : Quickselect
input  : The samples $X$ and $k$
output: The kth OS
begin
    Select a pivot $p$
    $X_\leq \leftarrow \{X_i \in X | X_i \leq p\}$, $X_> \leftarrow \{X_i \in X | X_i > p\}$
    if $|W_\leq| > k$ then
        return Quickselect($W_\leq$, k)
    else if $|W_>| > k$ then
        return Quickselect($W_>$, $k - |W_\leq|$)
    else return $p$
end
```

The pivot selection is chosen as:

1. Median of a subset of size M_1
2. kth OS of a subset of size $M_2 = M_1 \frac{N_1}{N}$ where N_1 is the number of samples after the first partition step. Repeat this approach until problem is bounded
3. Use a linear combination of max and min points

B. REFERENCES

[1] J.L. Paredes and G.R. Arce, "Compressive Sensing Signal Reconstruction by Median Regression Estimates," *ICASSP Dallas, TX*, March 2010.

[2] G.R. Arce, *Nonlinear Signal Processing: A Statistical Approach*, John Wiley & Sons, Inc., 2005.

[3] C. Hoare, "Find (algorithm 65)," *Commun. ACM*, pp. 4:321–322, 1961.

[4] C. Martínez and S. Roura, "Optimal sampling strategies in quicksort and quickselect," *SIAM Journal on Computing*, vol. 31, no. 3, pp. 683–705, 2002.

[5] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81, 2009.

[6] R.W. Floyd and R.L. Rivest, "Expected time bounds for selection," *Commun. ACM*, vol. 18, no. 3, pp. 165–172, 1975.