Двумерный пиковый фильтр за почти линейное время

А.М. Белов[a], А.В. Веричев[a]

[a] *Самарский национальный исследовательский университет имени академика С.П. Королёва, 443086, Московское шоссе, 34, Самара, Россия*

Аннотация

Данная работа посвящена разработке быстрого алгоритма локальной пиковой фильтрации двумерных массивов. Продемонстрирована неоднозначность понятия локальный пик и введены дополнительные условия, ее устраняющие. Разработан корректный пиковый фильтр, учитывающий указанные условия. Для сравнительного анализа эффективности предложенного алгоритма описаны два известных алгоритма нахождения локальных максимумов. Приведены оценки вычислительной сложности алгоритмов для лучшего и худшего случаев, произведён анализ зависимости времени работы от размеров изображения и скользящего окна, а также количества локальных максимумов. Результаты экспериментальных исследований свидетельствуют о том, что корректный пиковый фильтр работает лучше своих некорректных аналогов.

Ключевые слова: локальный пик; пиковый фильтр; двумерный массив; цифровое изображение;

1. Введение

Локальной пиковой фильтрацией называется поиск локальных максимумов некоторой функции, представленной в виде массива её значений. В качестве базовой операции, пиковая фильтрация применяется в многочисленных методах обработки изображений, в том числе в нелинейной фильтрации изображений [1] и нахождении особых точек - для подавления не максимальных значений функции выраженности угла [2-5] или локализации точек на пирамиде разрешений [6-8]. Однако, несмотря на широкую распространённость задачи поиска локальных максимумов, понятие локальный пик достаточно трудно определить однозначно.

Пусть локальный пик есть максимальный элемент множества элементов массива, содержащихся в некотором прямоугольном окне. Предположим, что в локальную область попадают несколько максимальных элементов, например, фильтрации подвергается изображение шахматного поля. Тогда возможны два варианта:

- Пиками считаются *все* максимальные значения. В таком случае результат пиковой фильтрации изображения, состоящего из широких константных полей, будет совпадать с самим изображением, т.е. никакой фильтрации на самом деле осуществлено не будет.
- Не считать локальным пиком *ни одно* из максимальных значений. В этом случае результат пиковой фильтрации изображения зачастую будет состоять из малого количества пиков, что не приемлемо во многих приложениях.

Помимо указанных трудностей общего характера, при решении конкретной задачи могут возникать дополнительные ограничения, обусловленные спецификой предметной области.

Областью строгих максимумов назовём подобласть изображения с одинаковыми значениями яркости, полностью окружённую элементами с меньшими значениями. Аналогично, *областью нестрогих максимумов* назовём подобласть изображения с одинаковыми значениями яркости, окружённую элементами с меньшими значениями по вертикали или горизонтали. Под *строгим пиком* будем понимать единственный элемент некоторой области строгих максимумов. *М-пиком* назовём элемент области нестрогих максимумов, все соседние элементы которого в прямоугольном окне размером $M \times M$ не превышают значения этого элемента изображения. Указанную окрестность назовём *окрестностью М-пика*. Положение М-пиков в общем случае определено неоднозначно.

Пусть имеется некоторый список М-пиков изображения. Назовём *М-покрытием* соответствующего списка множество элементов изображения, попавших в окрестности М-пиков из списка. Список М-пиков изображения назовём ***корректным***, если каждая из существующих на изображении областей строгих максимумов имеет непустое пересечение с М-покрытием этого списка. Нетрудно заметить, что если на изображении встречаются только строгие пики, то корректность списка М-пиков эквивалентна попаданию всех таких пиков в покрытие списка.

Алгоритм нахождения М-пиков (далее - *алгоритм пиковой фильтрации*) назовём *корректным*, если результатом его выполнения является корректный список М-пиков для любого изображения. В случае обработки изображения, не все М-пики которого являются строгими пиками, стандартные алгоритмы пиковой фильтрации (наподобие представленных в разделе 2) не являются корректными.

В данной работе представлен корректный во введённом выше смысле алгоритм локальной пиковой фильтрации. Раздел 2 посвящен описанию двух стандартных алгоритмов пиковой фильтрации, используемых далее для сравнения эффективности предлагаемого алгоритма, описанного в разделе 3. Помимо описаний в соответствующих разделах даны теоретические оценки эффективности алгоритмов. Раздел 4 посвящён результатам экспериментального исследования эффективности предлагаемого алгоритма. Выводы и итоги подведены в заключении.

2. Вспомогательные алгоритмы пиковой фильтрации

В данном разделе представлены алгоритмы пиковой фильтрации, используемые в исследовании эффективности работы предлагаемого корректного пикового фильтра. Эти алгоритмы не являются корректными в описанном во введении смысле, однако, во-первых, способны находить уникальные пики в локальной области, которые корректный

фильтр также должен находить, и, во-вторых, их вычислительная сложность достаточно просто посчитать. Таким образом, описываемые алгоритмы предоставляют хорошую базу для сравнения.

2.1. Фильтрация прямым перебором

В методе фильтрации прямым перебором локальные пики находятся путём прямого сравнения значений текущего элемента со значениями *всех* элементов, попадающих в прямоугольное окно. Пиком считается элемент, значение которого строго больше значения любого соседнего элемента.

Алгоритм М (*Фильтрация прямым перебором*). На вход подаётся массив $I[i,j]$, $0 \le i < N_1$, $0 \le j < N_2$, окно задаётся прямоугольной областью размером $M_1 \times M_2$. На выходе алгоритм возвращает список P пар координат (i,j), соответствующих положениям локальных пиков в массиве $I[i,j]$.

М1. [*Итерация по элементам*] Для всех элементов матрицы $I[i,j]$, $0 \le i < N_1$, $0 \le j < N_2$, выполнить шаг М2.

М2. [*Обработка элемента*] Если $I[k,l] < I[i,j]$ для всех $max(i - \lfloor M_1 / 2 \rfloor, 0) \le k \le min(i + \lfloor M_1 / 2 \rfloor, N_1 - 1)$, $k \ne i$, $max(j - \lfloor M_2 / 2 \rfloor, 0) \le l \le min(j + \lfloor M_2 / 2 \rfloor, N_2 - 1)$, добавить координаты пика (i,j) в список P.

М3. [*Конец процесса*] Прекращение работы алгоритма. Список P содержит все уникальные локальные пики массива $I[i,j]$, найденные в окне размером $M_1 \times M_2$.

Для проверки условия для каждого элемента массива $I[i,j]$ требуется осуществить $M_1 M_2 - 1$ сравнение значения текущего элемента с соседями. Исключение составляют элементы массива, близкие к границам, для которых количество требуемых сравнений несколько меньше, однако их влияние невелико. Таким образом, вычислительная сложность алгоритма М составляет

$$(M_1 M_2 - 1) \cdot N_1 N_2 \qquad (1)$$

сравнений.

2.2. Фильтрация методом одномерных приближений

В методе одномерных приближений пиком считается элемент, значение которого строго больше значения любого соседнего элемента. На первом этапе с помощью алгоритма Л формируются два списка локальных пиков, найденных по строкам и по столбцам матрицы, т.е. в одномерных массивах. На втором этапе найденные списки пересекаются и оставшееся [чаще всего небольшое] количество пиков проверяется методом прямого перебора. Ускорение на первом этапе достигается за счёт использования вспомогательных переменных, позволяющих находить пики за один проход по одномерным массивам, а также за счёт пересечения списков на втором этапе, что оставляет лишь небольшое количество потенциальных кандидатов, для которых необходимо проверить выполнение условия максимальности значения элемента в двумерном окне.

Алгоритм Л (Фильтрация одномерного массива). Алгоритм реализует обработку одномерного массива скользящим окном. На вход подаётся массив $C[i]$, $0 \le i < N$, и размер одномерного окна M. Алгоритм использует две вспомогательные переменные MV и $Count$ для хранения текущего максимального значения в окне и его кратности, т.е. количества раз, которое это максимальное значение встречается в текущем окне. Обработка очередного элемента массива заключается в сравнении обрабатываемого элемента с текущим максимумом в проверки его уникальности – выполнении равенства $Count = 1$. На каждом шаге осуществляются проверки того, является ли элемент, добавляемый в окно справа, новым максимумом, и требуется ли увеличить кратность текущего максимума при равенстве этих значений. Также проверяется равенство значения элемента, выталкиваемого из окна слева, текущему максимуму, что требует уменьшения кратности и поиска нового максимума в окне если $Count = 0$. На выходе алгоритм возвращает список P координат (i), соответствующих положениям локальных пиков в массиве $C[i]$.

Л1. [*Инициализация*] Установить $MV \leftarrow 0$ и $Count \leftarrow 0$.

Л2. [*Итерация по элементам*] Для $0 \le i < N$ выполнить шаги Л3-Л6.

Л3. [*Проверка выталкиваемого элемента*] Если $i \ge \lfloor M / 2 \rfloor - 1$ и $C[i - \lfloor M / 2 \rfloor - 1] = MV$, установить $Count \leftarrow Count - 1$.

Л4. [*Поиск максимального элемента*] Если $Count = 0$, среди значений $C[j]$ для $max(i - \lfloor M / 2 \rfloor, 0) \le j < min(i + \lfloor M / 2 \rfloor, N - 1)$ найти максимальное и его кратность, сохранить значения в переменные MV и $Count$, соответственно.

Л5. [*Проверка добавляемого элемента*] Если $i < N - \lfloor M / 2 \rfloor$ и $C[i + \lfloor M / 2 \rfloor] > MV$, установить $MV \leftarrow C[i + \lfloor M / 2 \rfloor]$ и $Count \leftarrow 1$; иначе, если $i < N - \lfloor M / 2 \rfloor$ и $C[i + \lfloor M / 2 \rfloor] = MV$, установить $Count \leftarrow Count + 1$.

Л6. [*Проверка текущего элемента*] Если $C[i] = MV$ и $Count = 1$, то $C[i]$ - локальный пик, а значит добавить координаты пика (i) в список P.

Л7. [*Конец процесса*] Прекращение работы алгоритма. Список P содержит все уникальные локальные пики массива $C[i]$, найденные в окне размером M.

Лучшим случаем для алгоритма Л является входной массив, в котором увеличивающиеся локальные пики расположены на расстоянии $\lfloor M / 2 \rfloor$ друг от друга, т.е. новый максимум попадает в окно до того, как выталкивается текущий максимум, следовательно, осуществлять поиск нового текущего максимума по окну не требуется. В этом случае алгоритм выполняет три сравнения значений массива – сравнение текущего значения с максимумом, а также

дополнительные сравнения значений выталкиваемого и добавляемого в окно элементов с текущим максимумом. Таким образом, сложность алгоритма составляет $3N$ сравнений элементов массива.

Худшим случаем для алгоритма Л является входной массив уменьшающихся значений. В этом случае на каждом шаге приходится осуществлять поиск нового текущего максимума. Следовательно, сложность в худшем случае составляет $(M+1)N$ сравнений элементов массива.

Алгоритм Л и алгоритм М позволяют решить задачу пиковой фильтрации методом одномерных приближений.

Алгоритм П (*Фильтрация методом одномерных приближений*). На вход подаётся массив $I[i,j]$, $0 \le i < N_1$, $0 \le j < N_2$, окно задаётся прямоугольной областью размером $M_1 \times M_2$. На выходе алгоритм возвращает список P пар координат (i,j), соответствующих положениям локальных пиков в массиве $I[i,j]$.

П1. [*Поиск пиков по строкам*] Для $0 \le i < N_1$ найти с помощью алгоритма Л список P_i пиков в i-й строке $I[i,:]$.

П2. [*Список пиков по строкам*] Для $0 \le i < N_1$ из всех элементов j списка P_i сформировать пары координат (i,j) и добавить в общий список P_r.

П3. [*Поиск пиков по столбцам*] Для $0 \le j < N_2$, найти с помощью алгоритма Л список P_j пиков в j-м столбце $I[:,j]$.

П4. [*Список пиков по столбцам*] Для $0 \le j < N_2$, из всех элементов i списка P_j сформировать пары координат (i,j) и добавить в общий список P_c.

П5. [*Пересечение списков*] Найти пересечение P списков P_r и P_c.

П6. [*Фильтрация прямым перебором*] Обработать оставшиеся пики из списка P с помощью метода прямого перебора и удалить те, которые не являются пиками в двумерном окне.

П7. [*Конец процесса*] Прекращение работы алгоритма. Список P содержит все уникальные локальные пики массива $I[i,j]$, найденные в окне размером $M_1 \times M_2$.

Обозначим L_1 количество пиков по строкам в одномерном окне размером M_2, L_2 - количество пиков по столбцам в одномерном окне размером M_1, $L = max(L_1, L_2)$. Количество пиков, являющихся пиками одновременно по строкам и по столбцам, обозначим T, $T \le min(L_1, L_2)$.

Сложность поиска списков пиков по строкам и столбцам составляет $N_1 \cdot 3N_2$ и $N_2 \cdot 3N_1$ в лучшем случае, в худшем случае $N_1 \cdot (M_2+1)N_2$ и $N_2 \cdot (M_1+1)N_1$. Сложность пересечения двух списков равна в лучшем случае $O\big(L \cdot log(L)\big)$ сравнений, в худшем случае $O(L^2)$ сравнений [9]. Сложность шага фильтрации прямым перебором равна $(M_1M_2-1)T$. Следовательно, алгоритм П требует

$$O(3N_1N_2 + L \cdot log(L) + (M_1M_2-1)T) \quad (2)$$

сравнений элементов массива в лучшем случае и

$$O\big((M_1+M_2+2)N_1N_2 + L^2 + (M_1M_2-1)T\big) \quad (3)$$

сравнений элементов массива в худшем случае.

3. Алгоритм корректной локальной пиковой фильтрации

Предлагаемый алгоритм корректной локальной пиковой фильтрации основан на алгоритме пиковой фильтрации одномерного массива при помощи конечного автомата (алгоритм А). Для разрешения проблем, описанных во введении, предложено использовать конечный автомат с буфером. Описанный в данном разделе вариант реализации корректного пикового фильтра предполагает выполнение последовательного поиска пиков по строкам и последующего двумерного прореживания списка локальных пиков. Стоит отметить, что возможны и другие реализации: так, например, для обеспечения четырехсвязности найденных пиков алгоритм может быть дополнен этапом поиска пиков по столбцам, а для обеспечения восьмисвязности - этапами поиска пиков по диагоналям.

3.1. Фильтрация одномерного массива при помощи конечного автомата

Максимум – это локальная характеристика, однако описанные во введении ситуации не позволяют алгоритмам М и П находить все максимумы, и соответственно считать их корректными. Использование конечного автомата с буфером позволяет принять решение относительно тех отсчетов, решение для которых в момент их обработки принято быть не может, например, в случае присутствия в исходном массиве протяженных константных областей.

Пусть дан одномерный массив $C[i]$. Рассмотрим элементы $C_{-1} = C[i-1]$, $C_0 = C[i]$ и $C_1 = C[i+1]$ для некоторого i – элементы, попадающие в окно шириной длиной $M = 3$. Все девять возможных вариантов взаимной упорядоченности элементов в рассмотренном окне вместе с соответствующими состояниями конечного автомата приведены в таблице 1.

В таблице 1 использованы следующие понятия.

Классы отсчетов – максимум (W_1) или «не максимум» (W_0).

Буфер – последовательность отсчетов, решение относительно класса которых в настоящий момент принято быть не может.

Завершение буфера – процесс отнесения всех отсчетов буфера к одному из двух классов (синоним – классификация буфера). Правило отнесения представлено ниже.

Класс буфера – класс, к которому относятся все отсчеты в последовательности, относящиеся к буферу.

Класс старта и класс завершения буфера – W_0 и W_1.

Таблица 1. Варианты взаимной упорядоченности элементов массива в окне длиной 3

Состояние	Тип	Значение на выходе	Завершение буфера / класс завершения буфера	Старт буфера / класс старта буфера
$C_{-1} < C_0 > C_1$	максимум	C_0	*N/A* / *N/A*	*нет* / *N/A*
$C_{-1} > C_0 < C_1$	минимум	C_*	*N/A* / *N/A*	*нет* / *N/A*
$C_{-1} > C_0 > C_1$	убывание	C_*	*нет* / *N/A*	*нет* / *N/A*
$C_{-1} < C_0 < C_1$	возрастание	C_*	*нет* / *N/A*	*нет* / *N/A*
$C_{-1} = C_0 = C_1$	константа	класс буфера	*нет* / *N/A*	*нет* / *N/A*
$C_{-1} = C_0 > C_1$	переход к убыванию	класс буфера	*да* / W_1	*нет* / *N/A*
$C_{-1} = C_0 < C_1$	переход к возрастанию	класс буфера	*да* / W_0	*нет* / *N/A*
$C_{-1} > C_0 = C_1$	завершение убывания	класс буфера	*нет* / *N/A*	*да* / W_0
$C_{-1} < C_0 = C_1$	завершение возрастания	класс буфера	*нет* / *N/A*	*да* / W_1

N/A – не применимо, ситуация не может произойти.

Правило классификации буфера:

- Если классы старта и завершения буфера одновременно W_1, то класс буфера - W_1.
- В противном случае класс буфера – W_0.

Если буфер имеет результирующий класс W_1, то в качестве пиков, в зависимости от поставленной задачи, можно выбрать: центральный элемент буфера, все элементы буфера, элементы буфера, отстоящие на заданное расстояние и т.д.

Результатом работы описанного автомата является список пиков, обнаруженных в окне размера 3. Этот список необходимо проредить, в соответствии с заданным размером окна обработки. Для этого список сортируется по убыванию значений соответствующих элементов массива, после чего выполняется процедура прореживания (алгоритм ОП). Более подробное описание алгоритма пиковой фильтрации с помощью конечного автомата приведено далее.

Алгоритм А (*Фильтрация конечным автоматом*). Алгоритм реализует обработку одномерного массива скользящим окном. На вход подаётся массив $C[i]$, $0 \le i < N$, и размер одномерного окна M. На выходе алгоритм возвращает список локальных пиков $P = \{(c_p, i_p)\},\ 0 \le p < L,\ c_p \in C[i],\ i_p \in [0, N-1]$, с указанием их значений и положений массиве $C[i]$.

А1. [*Поиск пиков в окне длиной 3*] С помощью конечного автомата найти пики в массиве $C[i]$ и добавить соответствующие пары (c, i) в список P.

А2. [*Подготовка к прореживанию*] Отсортировать список пиков P по убыванию значений c.

А3. [*Прореживание пиков*] Обработать список пиков P с помощью алгоритма ОП.

А4. [*Конец процесса*] Прекращение работы алгоритма. Список P содержит все локальные пики массива $C[i]$, найденные в окне размером M.

Алгоритм ОП (*Одномерное прореживание списка пиков*). На вход подаётся упорядоченный по убыванию значений c_l список пиков $P = \{(c_p, i_p)\},\ 0 \le p < L,\ c_p \in C[i],\ i_p \in [0, N-1]$, и размер одномерного окна M. Алгоритм использует временный массив $\hat{C}[i] \ne \hat{c}$, $0 \le i < N$ для хранения «зоны влияния» обработанных пиков, где $\hat{c}$ – специальное значение, которым будет помечаться «зона влияния» обрабатываемого пика. На выходе алгоритм возвращает прореженный список пиков $\hat{P} = \{(c_p, i_p)\}, 0 \le p < \hat{L} \le L$.

ОП1. [*Итерация по элементам списка*] Для всех элементов $(c_p, i_p),\ 0 \le l < L$ списка P выполнить шаг ОП2.

ОП2. [*Обработка элемента*] Если $\hat{C}[i_p] \ne \hat{c}$ то добавляем элемент (c_p, i_p) в результирующий список $\hat{P}$ и для всех $max(i_p - \lfloor M/2 \rfloor, 0) \le k \le min(i_p + \lfloor M/2 \rfloor, N-1), k \ne i_p$ выполняем $\hat{C}[k] = \hat{c}$.

ОП3. [*Конец процесса*] Прекращение работы алгоритма. Список $\hat{P}$ содержит все локальные пики массива $C[i]$, найденные в окне размером M.

Обозначим L количество пиков в одномерном массиве. Нахождение списка локальных пиков с помощью конечного автомата требует $2N$ сравнений элементов массива. Сложность сортировки списка равна в лучшем случае $O(L \cdot log(L))$ сравнений, в худшем случае $O(L^2)$ сравнений [9]. Сложность шага одномерного прореживания списка пиков равна L. Следовательно, в лучшем случае алгоритм А требует $O(2N + L \cdot log(L) + L) = O(2N + L \cdot log(L))$ сравнений. В худшем случае алгоритм выполняет $O((2N + L^2 + L)) = O(2N + L^2)$ сравнений элементов массива.

3.2. Описание алгоритма

При выполнении пиковой фильтрации на двумерном массиве данных для каждой строки выполняется описанная ранее фильтрация с помощью конечного автомата, после чего списки локальных пиков, найденные по всем строкам,

объединяются в один результирующий. Далее этот список так же сортируется по убыванию значений соответствующих элементов массива и подвергается процедуре прореживания, но уже в соответствии с заданным двумерным окном обработки (алгоритм ДП). Более подробное описание алгоритма корректной пиковой фильтрации приведено далее.

Алгоритм К (*Корректная пиковая фильтрация*). На вход подаётся массив $I[i,j]$, $0 \le i < N_1$, $0 \le j < N_2$, окно обработки задаётся прямоугольной областью размером $M_1 \times M_2$. Алгоритм использует два временных списка P_r и P_c, для хранения результатов построчной и постолбцовой обработки. На выходе алгоритм возвращает список $P = \{(i_p, j_p)\}$, $0 \le p < L$, $i_p \in [0, N_1 - 1]$, $j_l \in [0, N_2 - 1]$ пар координат, соответствующих положениям локальных пиков в массиве $I[i,j]$.

К1. [*Поиск пиков по строкам*] Для $0 \le i < N_1$ найти с помощью алгоритма А список $P_i = \{(c,j)\}$ пиков в i-й строке $I[i,:]$ и добавить в общий список P соответствующие найденным пикам объекты (c,i,j).

К2. [*Подготовка к прореживанию*] Отсортировать список пиков P по убыванию значений c.

К3. [*Прореживание пиков*] Обработать список пиков P с помощью алгоритма ДП.

К4. [*Конец процесса*] Прекращение работы алгоритма. Список P содержит все уникальные локальные пики массива $I[i,j]$, найденные в окне размером $M_1 \times M_2$.

Алгоритм ДП (*Двумерное прореживание списка пиков*). На вход подаётся упорядоченный по убыванию значений c_p список пиков $P = \{(c_p, i_p, j_p)\}$, $0 \le p < L$, $c_p \in I[i,j]$, $i_p \in [0, N_1 - 1]$, $j_l \in [0, N_2 - 1]$, окно задаётся прямоугольной областью размером $M_1 \times M_2$. Алгоритм использует временный массив $\hat{C}[i,j] \ne \hat{c}$, $0 \le i < N_1$, $0 \le j < N_2$, для хранения «зоны влияния» обработанных пиков, где $\hat{c}$ – специальное значение, которым будет помечаться «зона влияния» обрабатываемого пика. На выходе алгоритм возвращает прореженный список координат пиков $\hat{P} = \{(i_p, j_p)\}$, $0 \le p < \hat{L} \le L$.

ДП1. [*Итерация по элементам списка*] Для всех элементов (c_p, i_p, j_p), $0 \le p < L$ списка P выполнить шаг ДП2.

ДП2. [*Обработка элемента*] Если $\hat{C}[i_p, j_p] \ne \hat{c}$ то добавляем элемент (i_p, j_p) в результирующий список $\hat{P}$ и для всех $max(i_p - \lfloor M_1 / 2 \rfloor, 0) \le k \le min(i_p + \lfloor M_1 / 2 \rfloor, N_1 - 1)$, $k \ne i_p$, $max(j_p - \lfloor M_2 / 2 \rfloor, 0) \le l \le min(j_p + \lfloor M_2 / 2 \rfloor, N_2 - 1)$, $l \ne j_p$ выполняем $\hat{C}[k,l] = \hat{c}$.

ДП3. [*Конец процесса*] Прекращение работы алгоритма. Список $\hat{P}$ содержит все локальные пики массива $I[i,j]$, найденные в окне размером $M_1 \times M_2$..

Обозначим L максимальное количество пиков в строках массива $I[i,j]$. Тогда поиск списка пиков алгоритмом А имеет сложность $O(2N_1N_2 + N_1L \cdot log(L))$ сравнений в лучшем случае и $O(2N_1N_2 + N_1L^2)$ сравнений в худшем случае. Сложность сортировки общего списка пиков в лучшем случае равна $O(N_1L \cdot log(N_1L))$ сравнений, в худшем случае $O({N_1}^2L^2)$ [9]. Сложность шага прореживания списка пиков методом прямого перебора равна $O(N_1L)$.

Следовательно, алгоритм К требует

$$O(2N_1N_2 + N_1L \cdot log(L) + N_1L \cdot log(N_1L) + N_1L) = O(2N_1N_2 + N_1L \cdot log(N_1L)) \quad (4)$$

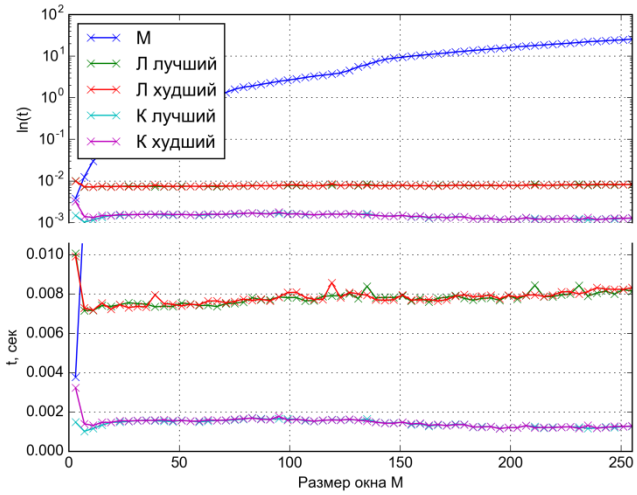
сравнений элементов массива в лучшем случае и

$$O(2N_1N_2 + N_1L^2 + {N_1}^2L^2 + N_1L) = O(2N_1N_2 + {N_1}^2L^2) \quad (5)$$

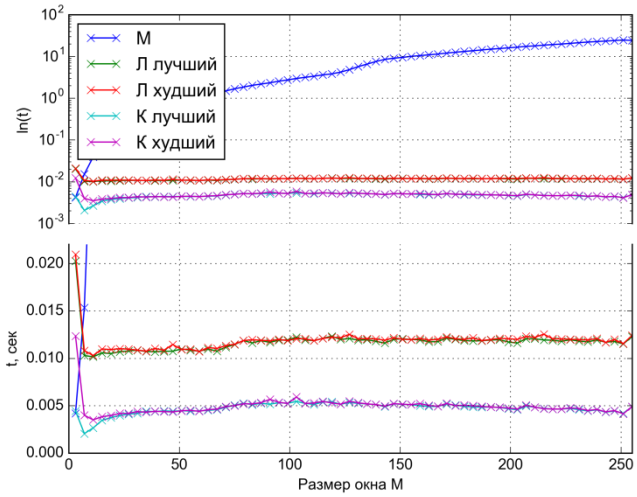
4. Экспериментальные исследования

Сравнение эффективности описанных в разделах 2 и 3 алгоритмов локальной пиковой фильтрации предлагается производить с помощью замеров процессорного времени, затрачиваемого на обработку изображений. Эксперименты производились с помощью ЭВМ на базе процессора Intel® Core™ i7-4770 3.40 ГГц с 16 ГБ оперативной памяти.

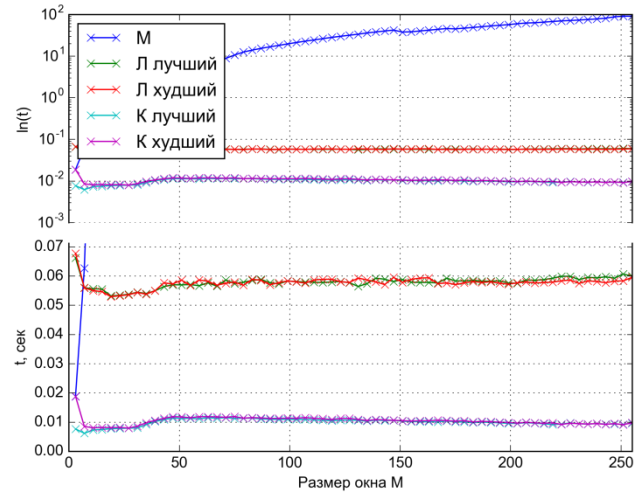
В эксперименте использовались синтезированные изображения размером $N \times N$ для значений $N = 512$ и $N = 1024$. Изображения инициализировались двумя способами. Во-первых, массивы заполнялись случайными значениями из диапазона $0..255$, при этом инициализация гарантировала равенство общего количества локальных пиков T части от максимально возможного значения $T_{max} = N_1N_1/((\lfloor M_1/2 \rfloor + 1) \cdot (\lfloor M_2/2 \rfloor + 1))$, $T = 0.25T_{max}$ и $T = T_{max}$. Во-вторых, инициализация имитировала худший случай для какого-либо алгоритма. Поиск локальных пиков осуществлялся в квадратном окне размером $M \times M$. Для каждой комбинации параметров производилось 25 замеров времени фильтрации изображения, в качестве результата бралось выборочное среднее полученных значений.



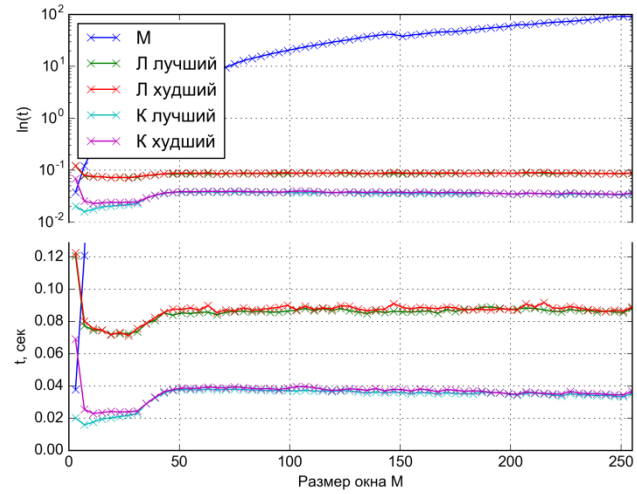
(а)



(б)



(в)



(г)

Рис. 1. Результаты исследования эффективности алгоритмов фильтрации изображений различных размеров с различным количеством пиков. (а) $N = 512$, $T = 0.25T_{max}$. (б) $N = 512$, $T = T_{max}$. (в) $N = 1024$, $T = 0.25T_{max}$.(г) $N = 1024$, $T = T_{max}$. Для детализации различий между быстрыми алгоритмами, результаты изображены на логарифмической (сверху) и обычной (снизу) шкале.

Подставим введённые ограничения (квадратные матрицы, квадратные окна) в формулы теоретической вычислительно сложности алгоритмов (1)-(5):

$$(M^2 - 1) \cdot N^2 \tag{6}$$

$$O(3N^2 + L \cdot log(L) + (M^2 - 1)T) \tag{7}$$

$$O(2(M + 1)N^2 + L^2 + (M^2 - 1)T) \tag{8}$$

$$O\big(2N^2 + NL \cdot log(NL)\big) \tag{9}$$

$$O(2N^2 + N^2L^2) \tag{10}$$

Анализ приведённых формул приводит к следующим выводам:

- Рост размера квадратного окна M увеличивает вычислительную сложность алгоритма К в меньшей степени, чем сложность чем алгоритма П. Алгоритм М существенно уступает им, причём отставание быстро увеличивается. Ср. рис. 1(а)-(г).

- Рост размера обрабатываемых изображений несколько сокращает разрыв в производительности между алгоритмом К и алгоритмом П, отставание алгоритма М становится ещё более ярко выраженным. Ср. рис. 1(а)-(б) и рис. 1(в)-(г).
- Рост количества локальных максимумов T в большей степени немного снижает эффективность алгоритма П, практически не влияя на алгоритм К и вовсе не имея никакого эффекта на алгоритм М. Ср. рис. 1(а) и рис. 1(б), рис. 1(в) и рис. 1(г).

Результаты проведённого эксперимента приведены на графиках, представленных на рис. 1. На графиках показано среднее время обработки изображений для различных размеров изображений, размеров окна, количества локальных пиков. Использованы алгоритмы:

- алгоритм М;
- алгоритм П на изображениях, представляющих лучший случай для этого алгоритма;
- алгоритм П в худшем случае;
- алгоритм К в лучшем случае;
- алгоритм К в худшем случае.

Полученные данные подтверждают выводы, сделанные из рассмотрения теоретических оценок сложности рассматриваемых алгоритмов.

5. Заключение

В работе введено понятие корректной пиковой фильтрации, предложен принцип построения алгоритмов корректной пиковой фильтрации. Проведен анализ эффективности предложенного алгоритма в сравнении с двумя известными алгоритмами поиска локальных максимумов. Результаты экспериментальных исследований показали, что корректный пиковый фильтр эффективнее своих некорректных аналогов. Так же стоит отметить, что структура предложенного алгоритма, такова, что допускает его параллельную реализацию без существенного изменения алгоритма.

Благодарности

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 17-29-03190-офи.

Литература

[1] Сойфер, В. А. Методы компьютерной обработки изображений / В. А. Сойфер. – М.: ФИЗМАТЛИТ, 2003. – 784 с.
[2] Förstner, W. A fast operator for detection and precise location of distinct points, corners and centres of circular features / W. Förstner, E. Gülch // Proc. ISPRS intercommission conference on fast processing of photogrammetric data - 1998. - P. 281-305.
[3] Harris, C. A combined corner and edge detector / C. Harris, M. Stephens // In Alvey vision conference - 1988. - Vol. 15(50). - P. 147-151.
[4] Shi, J. Good features to track / J. Shi, C. Tomasi // Proc. Intl Conf. on Comp. Vis. and Pat. Recog (CVPR) - 1994. - P. 593-600.
[5] Moravec, H. Rover visual obstacle avoidance / H. Moravec // Proc. Intl. Joint Conference on Artificial Intelligence. – 1981. – P. 785–790.
[6] Lowe, D. G. Object recognition from local scale-invariant features / D. G. Lowe // Proc. Intl. Conference on Computer Vision. – 1999. – P. 1150–1157.
[7] Bay, H. SURF: Speeded up robust features / H. Bay, A. Ess, T. Tuytelaars, L. Van Gool // Computer Vision and Image Understanding. – 2008. – V. 110. – P. 346–359.
[8] Lindeberg, T. Junction detection with automatic selection of detection scales and localization scales / T. Lindeberg // Proc. First Intl. Conference on Image Processing. –1994. – V.1. – P. 924–928.
[9] Кнут, Д. Э. Искусство программирования. Том 3. Сортировка и поиск / Д. Э. Кнут. – Москва: Вильямс, 2014. – 824 с.