УДК 517.98.537

СРАВНЕНИЕ ЧИСЛЕННЫХ РЕАЛИЗАЦИЙ АЛГОРИТМА РАСЧЁТА ВЗАИМНОЙ ИНФОРМАЦИИ НА ОСНОВЕ УЧЁТА БЛИЖАЙШИХ СОСЕДЕЙ

И. В. Сысоев

Национальный исследовательский
Саратовский государственный университет им. Н.Г. Чернышевского
Россия, 410012 Саратов, Астраханская, 83
E-mail: ivssci@gmail.com

Цель. Сравнить эффективность реализации различных подходов к оцениванию функции взаимной информации на основе учёта ближайших соседей.

Метод. Численно реализованы два подхода к вычислению функции взаимной информации: лобовой, основанный на поиске ближайших соседей перебором, и сортировочный, основанный на сортировке одного из наблюдаемых рядов.

Результаты. Показано, что алгоритмическая сложность сортировочного метода ниже, чем лобового, но выше, чем алгоритмическая сложность самой сортировки, реализованной любым из методов быстрой сортировки.

Обсуждение. Реализация сортировочного алгоритма оправдана в случае, если приходится иметь дело с выборками большой длины, в то время как для сравнительно небольших выборок (порядка сотен отсчётов) можно ограничиться лобовым подходом.

Ключевые слова: Взаимная информация, метод ближайших соседей, быстрая сортировка.

DOI:10.18500/0869-6632-2016-24-4-86-95

Ссылка на статью: Сысоев И.В. Сравнение численных реализаций алгоритма расчёта взаимной информации на основе учёта ближайших соседей // Известия вузов. Прикладная нелинейная динамика. 2016. Т. 24, № 4. С. 86–95.

Введение

Функция взаимной информации $I_{x,y}$ между двумя выборками $\{x_n\}_{n=1}^N$ и $\{y_n\}_{n=1}^N$ – популярная мера выявления связанности экспериментальных данных, широко используемая в задачах различных областей наук, в частности, в нелинейной динамике [1] для оптимальной реконструкции вектора состояния по скалярной временной реализации, в медицине для обработки объёмных изображений [2, 3], в нейрофизиологии [4] и даже в лингвистике [5]. Наиболее простой подход к расчёту функции взаимной информации – через оценку одномерных p_x и p_y и двумерного $p_{x,y}$ распределений с помощью гистограмм [6]. Его основной недостаток заключается в том, что для распределений с длинными хвостами, часто реализующихся на практике, необходим очень большой объём данных для получения более-менее достоверных оценок, иначе большое число редко населённых бинов приводит к большим погрешностям в оценке p_x, p_y и $p_{x,y}$, в результате чего оценка $I_{x,y}$ оказывается сильно ненадёжна.

Важность задачи породила целый ряд подходов [7], среди которых наиболее востребованными оказались основанные на использовании ядер [8] и учёте ближайших соседей [9]. Предложенный в [9] алгоритм имеет ряд важных преимуществ, в первую очередь с точки зрения достоверности получаемых оценок. Так, для линейных гауссовских процессов показано, что получаемые оценки асимптотически не смещённые с ростом длины выборки, а при отсутствии связи – и вовсе несмещённые при произвольной длине ряда, хотя для сигналов с более сложными свойствами, например, хаотических детерминированных, такие выкладки не проводились (и вряд ли в общем случае возможны). Кроме того, алгоритм имеет преимущество с вычислительной точки зрения: при оценивании расстояния между точками в двумерном пространстве используется не евклидова норма, а максимальный модуль разности по координатам (1), вычисление которого на ЭВМ гораздо проще и имеет большую точность, поскольку нет необходимости возводить в квадрат и брать квадратный корень.

$$d_{i,j} = \max(|x_i - x_j|, |y_i - y_j|). \tag{1}$$

Основная проблема лобовой реализации данного подхода лежит в вычислительной плоскости: поиск расстояния от каждой точки до каждой требует $O(N^2)$ операций, что при больших размерах выборки, например, $\sim 10^4$, приведёт к вычислительной сложности, требующей при реализации на современном персональном компьютере уже зримого – порядка секунд, десятков и даже сотен секунд – времени. При обработке единичной реализации это не становится существенным затруднением. Но при попарной обработке большого числа записей, например, сигналов стандартной 32-х канальной ЭЭГ, либо при исследовании зависимости функции взаимной информации сигнала самого с собою со сдвигом во времени, что необходимо, например, для определения оптимального вложения [1], время расчётов может быстро достигнуть неприемлемых значений – часов и даже дней. Сократить его можно, используя ряд технических приёмов: параллелизацию расчётов, кеширование результатов промежуточных вычислений, за счёт выбора языка программирования и компилятора. Некоторые детали этих приёмов рассмотрены далее. Проблема в том, что даже реализовав все эти оптимизации, уменьшить время расчётов удастся в фиксированные несколько или два-три десятка раз, что полезно, но не принципиально.

Поэтому наиболее перспективным является разработка алгоритма, имеющего меньший, чем $O(N^2)$, порядок вычислительной сложности, как это было указано в оригинальной работе [9]. *Целью данной работы* является сравнение лобового метода расчёта и метода, основанного на сортировке экспериментальных данных, при этом важно понять, для какого объёма выборки реализация сортировочного алгоритма является оправданною. При этом значительная часть дополнительных оптимизаций также была проведена: в частности, было реализовано распараллеливание расчётов и также выбраны язык программирования и компилятор, обеспечивающие высокую эффективность итогового кода. Отметим, что обзор методов сортировок и его реализация не является предметом данной работы и будет здесь излишним. Автор не претендует на новизну и значимость в этом вопросе.

1. Численные алгоритмы расчёта

Рассмотрим простейший вариант алгоритма, предложенного в [9], основанный на учёте одного первого соседа. Итоговая формула для вычисления оценки функции взаимной информации имеет вид:

$$I_{x,y} = \psi(N) + \psi(1) - \langle \psi(n_x(i)+1) + \psi(n_y(i)+1) \rangle_i \,, \tag{2}$$

где N – длина выборки, $n_x(i)$ и $n_y(i)$ – число соседей i-той точки на плоскости (X, Y) таких, что расстояние до них по одной из координат: x или y соответственно меньше расстояния до ближайшего соседа $\varepsilon_i/2$, найденного по формуле (1), $\psi(n)$ – дигамма функция.

1.1. Прямой алгоритм. Блоксхема прямой, или «лобовой» реализации метода расчёта функции взаимной информации на основе учёта первого ближайшего соседа приведена на рис. 1.

Оптимизация представленного алгоритма по производительности должна быть ориентирована в первую очередь на уменьшение числа и упрощение действий, производимых $O(N^2)$ раз. Можно выделить два места в представленном на рис. 1 варианте реализации, где возможно сокращение числа действий:

1. уменьшение числа проверок на несовпадение индексов – блоков `i<>j`;
2. уменьшение числа вычислений модуля разности вида `|x[i]-x[j]|` и `|y[i]--y[j]|` при расчёте расстояний.

Ликвидировать проверки в первом вложенном цикле можно, разбив его на два: от 0 до i не включительно и от $(i+1)$ до N не включительно. Ликвидировать проверки во втором цикле можно тем же способом, но есть более эффективный подход: если проверку убрать вовсе, величины $n_x(i)$ и $n_y(i)$ будут всегда ровно на 1 больше, чем нужно, поскольку i-тая точка всегда является соседом сама для себя. Однако в итоговое выражение входят $\psi(n_x(i)+1)$ и $\psi(n_y(i)+1)$, таким образом, заменив в аргументе дигамма функции $(n_x(i)+1)$ на $n_x(i)$ и убрав проверку на совпадение индексов, получим искомое выражение.

Для уменьшения числа вычислений модуля разницы лучше всего было бы создать матрицы расстояний между всеми точками по x и по y, тогда вместо четырёх вычислений величины $|x_i - x_j|$: двух на этапе расчёта $\varepsilon(i)/2$ и двух на этапе расчёта

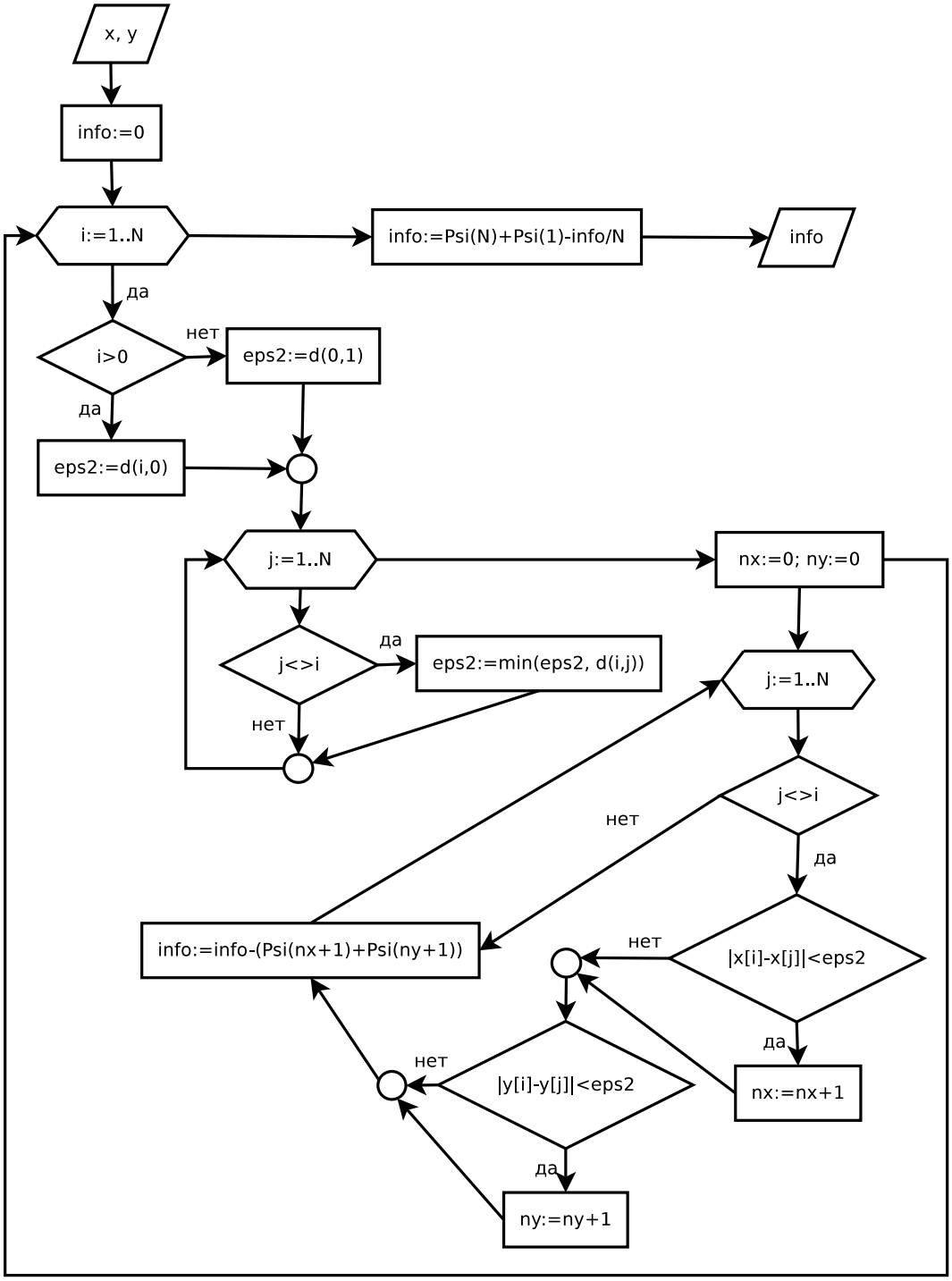

Рис. 1. Блок-схема прямого не оптимизированного алгоритма. Для краткости часть кода вынесена во встроенные функции: `max` и `min` – функции поиска максимума и минимума от 2 аргументов, `Psi` – дигамма функция, `d` – расстояние, рассчитанное по формуле (1)

$n_x(i)$ ограничиться только одним; аналогично и для $|y_i - y_j|$. Однако при больших размерах выборки это неприемлемо, так как размер матрицы будет расти пропорционально $O(N^2)$ и, например, для выборки размером в 10^4 значений составит уже 4999×10^4 элементов (учитывая, что матрица симметрична и хранение главной диагонали не требуется), что требует $\sim$ 200 Мб оперативной памяти при одинарной точности и $\sim$ 400 Мб при двойной, причём таких матриц придётся хранить две: для x_i и y_i. Понятно, что уже для 20–30 тысяч значений потребление памяти будет слишком велико даже, если проводить вычисления в числах с одинарною точностью.

Альтернативою созданию матрицы расстояний является хранение массивов значений $|x_i - x_j|$ и $|y_i - y_j|$ внутри цикла по i, хотя экономия в этом случае будет не столь велика, так как модуль разницы между $|x_p|$ и $|x_q|$ будет рассчитан дважды: при $p = i$, $q = j$ и при $p = j$, $q = i$. К сожалению, такой подход оказывается далеко не всегда оправдан, поскольку частично вступает в конфликт с другими оптимизациями. Основная проблема состоит в том, что заполнение массива является затратною с точки зрения числа обращений к памяти операцией, которая происходит $O(N^2)$ раз, так что вместо четырёх обращений к памяти на чтение за x_i и x_j получаем 2 обращения к памяти на чтение за x_i и x_j, одно на запись для расстояния $|x_i - x_j|$ и одно на чтения для него же.

Повысить производительность можно, кешировав значение x_i и y_i во внешнем цикле в локальные переменные. Эта оптимизация, однако, уменьшает эффект от предыдущей.

Ещё одним способом немного сократить время вычислений является оптимизация подхода к вычислению дигамма функции. Можно заметить, что нас интересуют только значения дигамма функции целого положительного аргумента, начиная с 1 и заканчивая N. Поэтому можно обратиться к итерационной формуле[1] из [10]:

$$\begin{aligned} &\psi(1) = -\gamma, \\ &\psi(n+1) = \psi(n) + 1/(n-1) \quad \forall n > 1, \end{aligned} \tag{3}$$

где γ – константа Эйлера[2]. Сделав всего N итераций в цикле, можно получить и сохранить в памяти все необходимые значения дигамма функции от всех аргументов от 1 до N, а затем вместо вычисления, например, $\psi(n_x(i))$ просто обращаться к $n_x(i)$-тому элементу сохранённого массива. Хотя число обращений к дигамма функции составляет всего $O(N)$, при не очень больших размерах выборки, например, в несколько сотен или тысяч точек, двухкратное вычисление сложной трансцедентной функции для произвольного действительного аргумента на каждом шаге может вести к существенным затратам вычислительных ресурсов.

Если описанных оптимизаций оказывается недостаточно, разумно задействовать возможности параллельных расчётов, что относительно просто реализуемо в данной ситуации благодаря фактической независимости операций внутри внешнего цикла по `i`.

Перечисленные оптимизации не могут решить главную проблему – вычислительная сложность сохраняет порядок $O(N^2)$.

[1] На самом деле итерационная формула работает для любых комплексных значений аргумента, не являющихся целыми отрицательными числами.

[2] $\gamma \approx 0.57721566490153286060651209008240243104 21$.

1.2. Сортировочный алгоритм. Практика показывает, что число ближайших соседей по каждой координате $n_x(i)$ и $n_y(i)$ очень невелико по сравнению с общею длиною выборки N и составляет порядка $\sqrt{N}$. Таким образом, если отсортировать исходные данные, например, в порядке возрастания величины x, то в новой выборке $\{x_{i'}\}_{i'=1}^{N}$ все ближайшие соседи по X точки с номером i' имеют близкие к ней номера. Точки с номерами $(i'+1)$ и $(i'-1)$ будут иметь в общем случае достаточно большое расстояние до точки с номером i' по координате Y, но если перебирать все точки с бóльшими и меньшими номерами последовательно, то довольно скоро (как раз на расстоянии $\sim \sqrt{N}$) можно будет найти точку с номером $(i' + l_{i'})$ (или

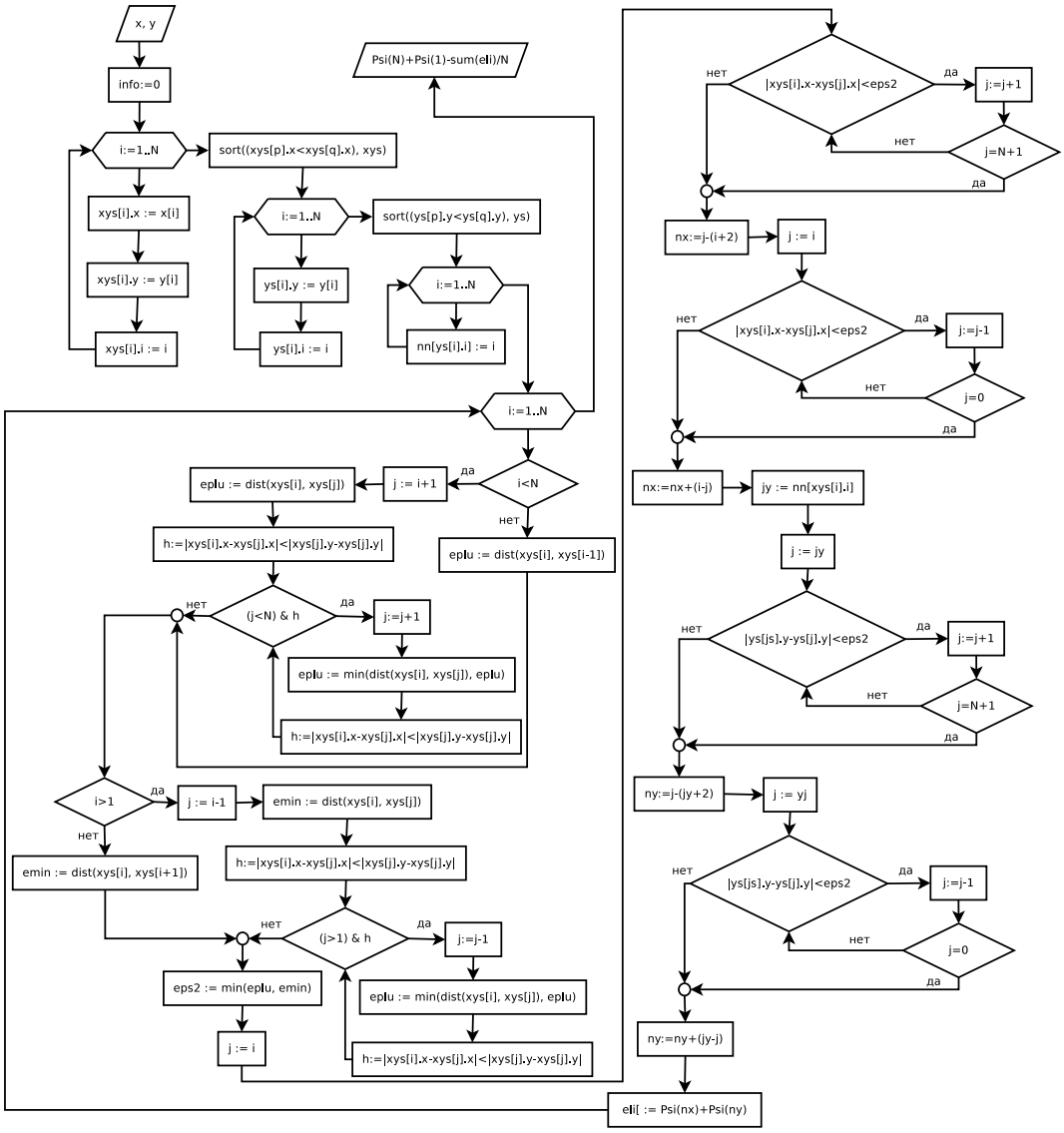

Рис. 2. Блоксхема сортировочного алгоритма. Для краткости часть кода вынесена во встроенные функции: `max` и `min` – функции поиска максимума и минимума от 2 аргументов, `Psi` – дигамма функция, `dist` – расстояние, рассчитанное по формуле (1), `sort` – функция сортировки записей, принимающая дополнительный аргумент – условие, `sum` – считает сумму элементов массива

$(i' - k_{i'})$ при переборе с уменьшением номера), для которой расстояние по Y окажется меньше, чем расстояние по X, после чего дальнейший перебор оказывается бессмысленным: согласно определению расстояния (1) более удалённые по X точки будут иметь большее расстояние до i-той, чем та, на которой перебор прекратился. То есть, поиск ближайшего соседа имеет смысл производить только среди элементов отсортированной выборки с номерами от $(i' - k_{i'})$ до $(i' + l_{i'})$. При этом, поиск ближайшего соседа и вычисление $k_{i'}$ и $l_{i'}$ разумно совместить.

Далее, отсортированный ряд $\{x_{i'}\}_{i'=1}^{N}$ легко использовать для определения числа $n_x(i)$, последовательно сравнивая расстояния между точками с близкими к i' номерами и i'-ой точкой до тех пор, пока оно не превысит $\varepsilon_i/2$. Для поиска соседей по Y нужно сделать аналогичную сортировку выборки в порядке возрастания величины Y: $\{y_{i''}\}_{i''=1}^{N}$. Кроме того, нужно при построении $\{x_{i'}\}$ сохранить номера, которые точки имели в исходной выборке, т. е. запомнить пары (i, i'). Тогда можно будет по номеру i' восстановить i. Также следует составить массив номеров i'', отсортированный в порядке увеличения соответствующих им номеров исходной выборки i. Тогда для определения i'', соответствующего i, достаточно просто обратиться к элементу массива с индексом i.

Поскольку все упомянутые сортировки можно произвести до основного цикла, а типичная быстрая сортировка имеет порядок сложности $O(N\log(N))$, их вклад в общее время работы программы оказывается незначительным. При этом сложность алгоритма в целом уменьшается с $O(N^2)$ до примерно $O(N\sqrt{N})$, считая, что $N\sqrt{N} \gg N\log(N)$, за счёт уменьшения числа итераций внутренних циклов. Блок-схема сортировочного алгоритма приведена на рис. 2.

2. Тестирование алгоритмов и выводы

Среднее по 20 запускам время работы прямого и сортировочного алгоритмов с учётом указанных выше оптимизаций приведено в таблице 1, где представлены однопоточный и многопоточный варианты. В качестве данных использовалась реализация двумерного гауссовского шума с нулевым средним и матрицею ковариаций $\begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$, где значение ρ варьировало в пределах от 0 до 1 (в таблице 1 приведены результаты для $\rho = 0.9$). Полученные в результате выполнения всех вариантов как прямого, так и сортировочного алгоритмов значения $I_{x,y}$ совпали и оказались близки к аналитически рассчитанному[3] значению $I_{x,y} = -\frac{1}{2}\ln\left(1-\rho^2\right)$.

	прямой, 1	прямой, 8	сорт., 1	сорт., 8
$< t >$, мс	13835	3158	94.43	35.27
σ_t, мс	150	215	1.78	1.89

Таблица 1. Среднее по 20 запускам время и среднеквадратичный разброс времени исполнения оптимизированного прямого и сортировочного алгоритма для однопоточной (помечено как 1) и многопоточной (помечено как 8) реализаций на процессоре AMD FX-8150. Программа реализована на языке D и скомпилирована компилятором gdc версии 4.8.1 с ключом оптимизации `-O2`. Многопоточная версия использует 8 потоков вычислений

[3]Среди прочего в [9] обсуждаются и погрешности предложенного метода, основанного на учёте ближайших соседей

Из таблицы 1 видно, что при использованной длине выборки в 2^{16} значений преимущество сортировочного алгоритма по скорости исполнения (время необходимое на операции ввода/вывода не учитывалось) составляет 146 раз для однопоточной реализации и 89 раз для многопоточной. Разница объясняется общим очень малым временем исполнения сортировочного метода, что снижает эффект от параллелизации алгоритма, поскольку затраты на создание и завершение дочерних потоков оказываются уже существенны. Кроме подробно проанализированной реализаций на языке D, алгоритм был также реализован на языках Pascal (диалект FreePascal, версия 2.6.4 с ключом -O2) и Fortran (использовался компилятор gfortran версии 4.8.1). В обоих случаях распараллеливание расчётов не проводилось. Версия на Fortran показала результаты очень близкие к однопоточной версии на D, что неудивительно, поскольку оба компилятора используют значительную часть общей кодовой базы набора компиляторов gcc, в том числе оптимизатор, причём использовалась одна и та же версия gcc. Программа на Psacal показала существенно худшие результаты: 50 с в прямой версии против 13.8 с для версии на D при $N = 2^{16}$, причём сортировочная версия была быстрее прямой примерно в 130 раз. Таким образом, как и ожидалось, ускорение, достигаемое за счёт применения сортировочной версии алгоритма, в целом не зависит от языка программирования, хотя абсолютное время расчётов может варьировать в несколько раз.

Чтобы охарактеризовать зависимость результатов от длины выборки, аналогичные вычисления были проведены для других длин, начиная от 512 – см. рис. 3.

Из рис. 3 видно, что в двойном логарифмическом масштабе зависимости близки к прямым, что и должно быть, поскольку для обоих алгоритмов теоретически эта зависимость носит степенной характер: $O(N^2)$ для прямого и $O(N\sqrt{(N)})$ для сортировочного. При этом абсолютная разница во времени работы между прямым и сортировочным методами увеличивается с ростом размера выборки, что соответствует предположению о том, что прямые, описывающие теоретически описывающие полученные результаты, должны иметь различный наклон – больший для прямого метода. Можно также отметить, что для относительно малых длин выборки – менее двух тысяч – дисперсия значений времени выполнения очень велика, что выражается в «неровности» получившихся кривых. При таких длинах существенный вклад во время исполнения сортировочного алгоритма дают сами сортировки, а для вариантов с распараллеливанием – затраты на создание, поддержание и завершение дочерних потоков.

Таким образом, по результатам проведённого сопоставления сортировочный алгоритм оказывается гораздо более эффективным, нежели лобовая реализация. Тем не менее, время выполнения лобовой реализации, особенно с учётом возможной параллелизации, оказывается приемлемым во многих случаях. Учитывая заметно большую сложность реализации сортировочного алгоритма, его преимущества оказываются существенными в случае длинных выборок и больших ансамблей, когда

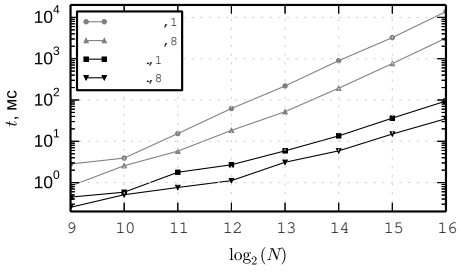

Рис. 3. Зависимость времени расчётов от длины выборки, усреднённая по 20 запускам. Различные кривые соответствуют прямому и сортировочному алгоритмам, выполняемым в 1 или 8 потоках

время написания программы начинает существенно уступать суммарному времени её выполнения.

Важно отметить, что сложность самого оптимизированного алгоритма составляет порядка $O(N\sqrt{(N)})$, то есть она заведомо выше, чем сложность любого известного метода быстрой сортировки, составляющая $O(N\ln(N))$. Таким образом, в принципе не важно, какой алгоритм быстрой сортировки будет использован, поскольку, хотя использование сортировки снижает сложность алгоритма поиска ближайших соседей, но снижает его недостаточно, чтобы сама сортировка стала узким местом алгоритма.

Оптимизированный с учётом сортировки алгоритм может быть адаптирован для более сложных мер таких, как энтропия переноса [11] и информации взаимодействия [12] – везде, где необходим оценка многомерных плотностей распределений по наблюдаемой выборке.

Работа выполнена при поддержке РФФИ, гранты № 14-02-00492, № 16-34-00203 и стипендии Президента РФ для молодых учёных СП-1510.2015.4.

Библиографический список

1. *Fraser A.M., Swinney H.L.* Independent coordinates for strange attractors from mutual information // Phys. Rev. A. 1986. Vol. 33. 1134.
2. *Wells W.M., Viola P., Atsumi H., Nakajima Sh., Kikinis R.* Multi-modal volume registration by maximization of mutual information // Medical Image Analysis. 1996. Vol. 1. Iss. 1. Pp. 35–51.
3. *Pluim J.P.W., Maintz J.B.A., Viergever M.A.* Mutual-information-based registration of medical images: a survey // IEEE Transac. on Medical Imaging, 2003. Vol. 22, Iss. 8. Pp. 986–1004.
4. *Paninski L.* Estimation of Entropy and Mutual Information // Neural Computation. 2003. Vol. 15. Pp. 1191–1253.
5. *Church K.W., Hanks P.* Word association norms, mutual information, and lexicography // Computational Linguistics. 1990. Vol. 16, Iss. 1. Pp. 22–29.
6. *Moddemeijer R.* On estimation of entropy and mutual information of continuous distributions // Signal Processing. 1989. Vol. 16, Iss. 3. Pp. 233–248.
7. *Ai-Hua Jiang, Xiu-Chang Huang, Zhen-Hua Zhang, Jun Li, Zhi-Yi Zhang, Hong-Xin Hua.* Mutual information algorithms // Mechanical Systems and Signal Processing. 2010. Vol. 24. Pp. 2947–2960.
8. *Il-Moon Yo., Rajagopalan B., Lall U.* // Estimation of mutual information using kernel density estimators // Phys. Rev. E. 1995. Vol. 52(3). Pp. 2318–2321.
9. *Kraskov A., Stögbauer H., Grassberger P.* Estimating mutual information // Phys. Rev. E. 2004. Vol. 69. 066138.
10. *Abramowitz M., Stegun I.A., eds.* Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables (10th ed.). New York: Dover. 1972. Pp. 258–259.
11. *Schreiber Th.* Measuring Information Transfer // Phys. Rev. Lett. 2000. Vol. 85. Pp. 461–464.
12. *McGill W.J.* Multivariate information transmission // Psychometrika. 1954. Vol. 19. Pp. 97–116.

Поступила в редакцию 10.08.2016

COMPARISON OF NUMERICAL REALISATION OF ALGORITHM OF MUTUAL INFORMATION CALCULATION BASED ON NEAREST NEIGHBOURS

I. V. Sysoev

National Research Saratov State University
Astrahanskaya, 83, 410012 Saratov, Russia
E-mail: ivssci@gmail.com

Purpose. To compare effeciency of different realizations of approaches to estimation of mutual information function based on nearest neighbours.

Method. Two approaches to calculation of mutual information function were realized numerically: straightforward approach is based on brute force, and sorting based one.

Results. The algorithmic complexity of sorting beased method was shown to be less than of straightforward approach, but larger than the complexity of any quick sort method.

Discussion. Realization of sorting based method is reasonable in the case, when one has to deal with long samplings, while for small samplings the straightforward approach is enough.

Keywords: Mutual information, nearest neighbours method, quick sort.

DOI: 10.18500/0869-6632-2016-24-4-86-95

Paper reference: Sysoev I.V. Comparison of numerical realisation of algorithm of mutual information calculation based on nearest neighbours // Izvestiya VUZ. Applied Nonlinear Dynamics. 2016. Vol. 24. Issue 4. P. 86–95.

Сысоев Илья Вячеславович – родился в Саратове в 1983 году. Окончил Саратовский государственный университет имени Н. Г. Чернышевского в 2004 году, защитил диссертацию на соискание учёной степени кандидата физико-математических наук в 2007 году. С 2005 года работал на кафедре электроники, колебаний и волн ассистентом. С 2008 года доцент кафедры динамического моделирования и биомедицинской инженерии СГУ.

Основные научные интересы: анализ временных рядов, в том числе разработка и адаптация к экспериментальным данным методов анализа связанности, изучение сигналов мозга, изучение эпилепсии.

410012 Саратов, Астраханская, 83
Саратовский государственный университет имени Н.Г. Чернышевского
E-mail: ivssci@gmail.com