Санкт-Петербургский государственный университет
**Кафедра моделирования электромеханических и компьютерных систем**

## Бабаева Ирина Витальевна

**Магистерская диссертация**

# Методы анализа информационной чувствительности компьютерных алгоритмов

Направление 01.04.02
«Прикладная математика и информатика»
Магистерская программа «Прикладные информационные технологии. Информационные экспертные системы»

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Никифоров К.А.

Санкт-Петербург
2018

## Содержание

# Введение

Одной из распространенных оценок эффективности компьютерных алгоритмов является временная оценка. Однако, временная оценка эффективности алгоритма представляет собой асимптотическую зависимость, показывающую рост функции длины входа, поэтому такая оценка объективна только на больших длинах входов. Чтобы иметь возможность сравнивать алгоритмы в реальных условиях, необходимо вычисление количества базовых операций алгоритмов, которые они выполняют, т. е. их функций трудоемкости.

При исследовании временной оценки эффективности алгоритма функция трудоемкости ассоциирована со временем получения результата на определенных исходных данных. В этом аспекте результаты ресурсного анализа алгоритма должны обеспечивать возможность прогнозирования его трудоемкости как для разных длин входов, так и для различных входов фиксированной длины.

При решении задачи прогнозирования на фиксированной длине входа количественно-зависимые алгоритмы (класс N [1]) представляют собой наиболее благоприятный класс. Но, к сожалению, подавляющее большинство алгоритмов относятся к количественно-параметрическому классу (класс NPR) и обладают ненулевым размахом варьирования трудоемкости при статичной длине входа. Для ряда алгоритмов на актуальных длинах входа значение этого размаха достаточно велико. Прогнозирование по трудоемкости в худшем случае (гарантированная оценка сверху) часто дает для этого класса завышенные результаты, а прогнозирование по трудоемкости в среднем случае не всегда учитывает информацию о размахе варьирования. При этом очевидно, что качество прогноза в основном определяется

тем, насколько велика зависимость трудоемкости от различных входов фиксированной длины.

Помимо необходимости обеспечения качества прогнозирования в некоторых случаях, для систем реального времени, возникает задача обеспечения стабильности по времени программной реализации алгоритма. Что означает слабую зависимость времени выполнения от исходных данных при фиксированной длине входа. Для решения задачи рационального выбора алгоритма с учетом этого требования также необходима детальная информация о влиянии на его трудоемкость различных входов фиксированной длины. При этом в качестве оценки стабильности по времени реализации компьютерного алгоритма может быть использована и количественная оценка его информационной чувствительности.

Основными критериями оценки алгоритма являются временная и емкостная сложности, т. е. оценки требований алгоритма к ресурсам процессора и оперативной памяти. Такие оценки необходимы, чтобы на этапе проектирования программной системы исключить превышение ограничений технического задания по памяти или временной эффективности, а также выявить узкие по ресурсоемкости места алгоритмического обеспечения.

## Постановка задачи

Целью работы является разработка и применение методов анализа информационной чувствительности компьютерных алгоритмов, которые давали бы более достоверную оценку при прогнозировании эффективности компьютерного алгоритма по сравнению с временной оценкой.

Для достижения поставленной цели необходимо решить такие подзадачи, как:

1. исследование уже существующих методов анализа алгоритмов;
2. переход от оценки по времени к оценке количества базовых операций;
3. обоснование возможности предложения нового метода.

Существующие методы классического анализа алгоритмов позволяют аналитически получить асимптотические оценки (их неудобство уже упоминалось выше), причем необходимо работать с самим алгоритмом, а не с программной реализацией, что ограничивает практическую применимость методов.

Существующие практические подходы по измерению времени работы программной реализации алгоритма выдают результаты, которые сильно привязаны к аппаратному и системному программному обеспечению вычислительного эксперимента. В данной работе необходимо разработать более универсальный подход, основанный на подсчете количества базовых операций, выполненных программой на выборке входных данных, с переносом результатов анализа программной реализации на оценку лежащего в ее основе алгоритма.

Таким образом, возникает задача определения набора базовых операций, в единицах которых измеряется трудоемкость.

Этот набор должен соответствовать базовым операциям, выполняемым на процессоре. В ходе выполнения данной работы будут рассмотрены два набора базовых операций:

- такты процессора;
- инструкции процессора;

В теоретических моделях, например, в машине Тьюринга, базовый набор соответствует правилам перехода блока управления. В более сложных моделях вычислений, например в гипотетической вычислительной машине MIX [2], система команд приближена к реальному процессору, для которого ассемблерный код программы представляет последовательность команд, каждая из которых (при упрощенном рассмотрении) выполняется за один такт.

Поэтому реализацию рассмотренных методов следует планировать с использованием средств профилирования программной реализации алгоритма, выбранного для анализа. Сам выбор алгоритма не должен играть большой роли, т. к. разрабатываемые методы должны быть достаточно универсальными, хотя используемый тип данных имеет значение — алгоритмы с целочисленными типами данных (алгоритмы задач дискретной оптимизации и т. п.) или алгоритмы с дробными типами с плавающей запятой, которые часто используются при научных вычислениях.

## Обзор литературы

Для того, чтобы решить описанные выше задачи, необходимо изучить уже существующие методы и подходы в области оценки информационной чувствительности и изучить возможные наборы базовых команд, в контексте которых может быть измерена функция трудоемкости.

Определение функции трудоемкости и понятие информационной чувствительности вводятся М. В. Ульяновым в [1]. Способы оценки информационной чувствительности различными способами, практическая польза и применимость таких оценок при разработке программных систем описываются в [3] и [4].

Для определения базового набора команд, необходимого для измерения функции трудоемкости, требуется изучить системы команд, приближенные к системе команд процессора. Для понимания общего устройства и работы процессора были использованы источники [2] и [13]. Также, для того, чтобы получить оценку кода и определить поведение алгоритма во время выполнения программы, необходимо рассмотреть средства профилирования и дизассемблирования, описанные в [6], [7], [8] и [12]. С целью провести анализ исходного кода алгоритма был рассмотрен Roslyn, описанный в [9] и [10].

Инструмент Intel Vtune позволяет подсчитывать базовые команды процессора, которые необходимы для выполнения алгоритма. Для изучения Intel Vtune были рассмотрены [15], [16], [17] и форум компании Intel.

Для проведения тестирования предложенного в данной работе метода была выбрана задача коммивояжера и пакет для ее решения Concorde TSP Solver, в состав которого входит QSopt. Описание задачи коммивояжера и упомянутых пакетов приво-

дится в [18], [19] и [20].

Таким образом, с помощью изучения описанных выше источников литературы, становится возможным решение задач, необходимых для достижения цели, сформулированной в предыдущем разделе.

## 1. Информационная чувствительность

Прежде, чем определить понятие информационной чувствительности, введем следующие обозначения:

$A$ — алгоритм

$D$ — вход алгоритма $A$: конечное множество слов фиксированной длины в бинарном алфавите, задающее конкретную решаемую задачу;

$D_A(x)$ — множество возможных конкретных экземпляров для задачи, решаемой с помощью алгоритма $A$ (множество допустимых входов алгоритма);

$f_A(D)$ — функция трудоемкости для входа $D$.

Обозначим различные случаи трудоемкости на всех входах фиксированной длины следующим образом:

$$f_A^{\wedge}(n) = max_{D \in D_n} \{f_A(D)\} \text{ — худший случай.}$$

Под худшим случаем понимается наибольшее количество операций, задаваемых алгоритмом A на всех входах размерности $n$.

$$f_A^{\vee}(n) = min_{D \in D_n} \{f_A(D)\} \text{ — лучший случай.}$$

Под лучшим — наименьшее количество операций, задаваемых алгоритмом A на всех входах размерности $n$.

$\overline{f_A}(n) = \sum_{D \in D_n} p(D) f_A(D)$ — средний случай трудоемкости.

Под средним — среднее количество операций, задаваемых алгоритмом A на всех входах размерности $n$.

$$f_A^{\vee}(n) \leq f_A(D \in D_n) \leq f_A^{\wedge}(n). \quad (1)$$

Прогнозирование по трудоёмкости в худшем случае (гарантированная оценка сверху) даёт почти всегда сильно завышенные результаты. Прогнозирование по трудоёмкости в среднем случае не учитывает информацию о размахе варьирования.

Качество прогноза во многом определяется влиянием различных входов фиксированной длины на трудоёмкость или информационной чувствительностью исследуемого алгоритма в рамках выбранной количественной оценки [3].

После определения трудоемкости, можем перейти к определению информационной чувствительности.

Информационная чувствительность — это влияние различных входов фиксированной размерности на изменение значений функции трудоемкости алгоритма [1].

Впервые понятие информационной чувствительности алгоритма по трудоемкости введено М.В. Ульяновым и В.А. Головешкиным. Понятие «информационная чувствительность» отражает тот факт, что алгоритм задает различное число базовых операций принятой модели вычислений $f_A(D)$ на разных входах $D$, имеющих фиксированную длину $n$. Ключевым для содержательной интерпретации этого термина является выбор количественной оценки (меры), обладающей свойством сопоставимости, т. е. дающей возможность решения задач сравнения алгоритмов и их рационального выбора [3].

## 1.1. Статистическая оценка информационной чувствительности

Статистическая оценка информационной чувствительности предложена М.В. Ульяновым и В.А. Головешкиным на основе следующих рассуждений. На множестве входов фиксированной длины трудоемкость алгоритма рассматривается как дискретная ограниченная случайная величина.

Классической точечной мерой рассеяния случайной величины является $\sigma$ — среднеквадратическое отклонение, которое оценивается по данным выборки через стандартное отклоне-

ние $s$. Определение количественной оценки информационной чувствительности должно учитывать также и длину сегмента варьирования. Это связано с тем, что при одинаковом значении дисперсии достоверно более чувствительным должен быть алгоритм с большей длиной сегмента возможных значений трудоемкости. Для учета длины этого сегмента используется такое понятие математической статистики, как размах варьирования.

Очевидно, что теоретический размах варьирования трудоемкости алгоритма является функцией длины входа, тогда, вводя обозначение $R(n)$ с учетом (1), получаем:

$$R(n) = f_A^{\wedge}(n) - f_A^{\vee}(n).$$

На основе этого вводится понятие нормированного (относительного) размаха варьирования функции трудоемкости для входов длины $n$ — $R_N(n)$ как отношение половины теоретического размаха варьирования к его середине

$$R_N(n) = \frac{f_A^{\wedge}(n) - f_A^{\vee}(n)}{f_A^{\wedge}(n) + f_A^{\vee}(n)}. \tag{2}$$

Одной из общепринятых точечных характеристик выборки является коэффициент вариации $V$, определяемый как отношение стандартного отклонения к среднему значению. Для выборки, полученной на основе экспериментального исследования трудоемкости алгоритма, коэффициент вариации $V$ также зависит от размерности и имеет вид:

$$V(n) = s_{f_A}(n) / \overline{f_A}(n), 0 \leq V(n) \leq 1, \tag{3}$$

где $s_{f_A}(n)$ — стандартное отклонение трудоемкости, как дискретной ограниченной случайной величины, при фиксированной длине входа $n$, а $\overline{f_A}(n)$ — выборочное среднее, рассчитываемые по данным выборки. На основе этих рассуждений вводится статистическая количественная оценка (мера) информа-

ционной чувствительности алгоритма по трудоемкости, с обозначением $\delta_{IS}(n)$, в виде:

$$\delta_{IS}(n) = V(n)R_N(n), 0 \leq \delta_{IS} \leq 1. \quad (4)$$

Поскольку оценка $\delta_{IS}(n)$ использует только статистические точечные оценки трудоемкости как случайной величины, ее применение возможно в случае отсутствия знаний о законе распределения значений трудоемкости или какой-либо его аппроксимации. Таким образом, значения оценки $\delta_{IS}(n)$ могут быть получены на основе экспериментальных исследований алгоритма, по результатам которых вычисляется значение $V(n)$, и его теоретического анализа, необходимого для вычисления нормированного размаха варьирования [4].

## 1.2. Оценка информационной чувствительности по энтропии

Классической характеристикой хаотичности некоторой системы является энтропия — понятие, впервые введенное Клаузиусом в термодинамике в 1865 г. для определения меры необратимого рассеивания энергии. В теорию информации энтропия введена К. Шенноном как мера случайности, мера неопределенности какого-либо испытания, имеющего разные исходы. Для дискретной случайной величины $X$, принимающей $n$ значений с вероятностями $p_i$, где $i = (1, n)$, энтропия $H(X)$ вычисляется по формуле:

$$H(X) = -\sum_{i=1}^{n} p_i log_2 p_i \quad (5)$$

и характеризует степень разнообразия состояний системы. Энтропия обращается в нуль лишь в том случае, если состояние системы полностью определено, и максимальна, когда

все состояния равновероятны, что легко доказывается методом неопределенных множителей Лагранжа. Значение $H(X)$ характеризует среднюю неопределенность выбора одного состояния из ансамбля и зависит, в силу (2), только от вероятностей состояний.

Для непрерывных случайных величин неопределенность значений состояния системы связана с плотностью распределения вероятностей этих значений — дифференциальным законом распределения $p(x)$. В связи с этим непрерывный аналог формулы (2) получил название относительной дифференциальной энтропии или просто дифференциальной энтропии:

$$h(X) = -\int_{-\infty}^{\infty} p\,(x)\, log_2 p(x) dx \qquad (6)$$

Значение $h(X)$ можно трактовать как среднюю неопределенность случайной величины $X$ с произвольным законом распределения по сравнению со средней неопределенностью случайной величины с размахом варьирования, равным единице при равномерном распределении. Дифференциальная энтропия так же не зависит от конкретных значений случайной величины $X$. Если $X$ является случайной величиной с ограниченной вариацией, то максимальная дифференциальная энтропия соответствует равномерной плотности распределения вероятностей.

Использование энтропийного подхода позволило получить ряд значимых результатов как в теории информации, так и целом ряде других областей, например, при анализе динамических систем.

Энтропийную оценку информационной чувствительности можно вычислить на основе экспериментальной гистограммы относительных частот трудоемкости по формуле (5).

Энтропийная оценка информационной чувствительности

позволяет качественно оценить информационную чувствительность алгоритма, не локализуя при этом положение сегмента значений с наибольшими вероятностями [4].

## 1.3. Квантильная оценка информационной чувствительности

Основная идея состоит в определении длины сегмента нормированных значений трудоемкости, по которому интеграл от функции плотности равен заданной вероятности (надежности). Отметим также, что эта оценка не содержит информацию о положении гамма-квантиля закона распределения на нормированном сегменте.

Квантильная количественная мера информационной чувствительности алгоритма $\delta_{IQ}(\gamma, n)$ (при фиксированной длине входа) есть длина симметричного относительно медианы сегмента значений трудоемкости алгоритма (рассматриваемых как реализации случайной величины и нормарованных к сегменту [0, 1] по теоретическим границам), по которому интеграл от аппроксимирующей функции плотности распределения вероятностей равен $\gamma$.

Если функция $F(n, x), x \in [0, 1]$, есть аппроксимирующая функция распределения значений трудоемкости, параметризированная по длине входа алгоритма $n$, а функция $F^{-1}(n, x)$ есть функция, обратная к $F(n, x)$, то

$$\delta_{IQ}(\gamma, n) = F^{-1}(n, \frac{1}{2} + \frac{\gamma}{2}) - F^{-1}(n, \frac{1}{2} - \frac{\gamma}{2}) \qquad (7)$$

Можно интерпретировать введенную меру $\delta_{IQ}(\gamma, n)$ как относительную длину доверительного интервала значений трудоемкости относительно медианы, в который трудоемкость алгоритма на произвольном входе попадает с вероятностью $\gamma$. Ин-

туитивно понятно, что с уменьшением разброса значений трудоемкости относительно медианы, приводящим к уменьшению дисперсии, функция плотности будет «сжиматься» к медиане, и информационная чувствительность алгоритма по трудоемкости будет падать.

Использование медианы, а не математического ожидания в определении квантильной меры информационной чувствительности связано с тем, что для несимметричных функций плотности распределения с ограниченной вариацией медиана и математическое ожидание не обязательно совпадают. Использование в этом случае математического ожидания как центра сегмента интегрирования функции плотности может привести, для близких к единице значений, к выходу одной из границ доверительного интервала за соответствующую границу сегмента определения функции плотности.

Количественная мера информационной чувствительности $\delta_{IQ}(\gamma, n)$, основанная на вычислении квантилей аппроксимирующей функции плотности, может использоваться при интервальном прогнозировании временной эффективности компьютерных алгоритмов и при решении задачи выбора рациональных алгоритмов по критерию временной устойчивости с учетом доверительной вероятности, заданной пользователем [4].

## 2. Способ измерения трудоемкости

Подходы оценки информационной чувствительности, описанные в Главе 1, опираются на временную оценку эффективности алгоритмов, что влечет за собой некоторую погрешность такой оценки. Погрешность возникает вследствие взаимодействия с процессором операционной системы и другого окружения, из-за чего часто приходится брать приближенную временную оценку. Для того, чтобы оценивать эффективность алгоритмов более объективно, можно рассмотреть задачу подсчета количества базовых операций, которые выполняются процессором во время работы алгоритма.

В данной работе в качестве базовых операций процессора были рассмотрены две величины:

- такты процессора;
- базовые инструкции процессора.

Один такт процессора соответствует одному периоду импульсов тактового генератора и является основной единицей измерения времени выполнения команд процессором. Система команд вычислительной машины, как уже упоминалось выше, может быть ассоциирована с ассемблерным кодом программы. В свою очередь, каждая команда, записанная на языке ассемблера, гарантированно выполняется за определенное количество тактов, которое указано в документации Intel [5].

Макроинструкции Intel, такие как ADD, MOV, SUB и т. д., разбиваются на ряд микроопераций (uops), где одна макроинструкция может содержать несколько микроопераций. Под инструкцией процессора в данной работе будет пониматься микрооперация: uop.

Таким образом, использование в качестве количественной

меры для функции трудоемкости как тактов процессора, так и микроинструкций, даст более объективную оценку, потому что базовые операции процессора, которые не относятся к исследуемому алгоритму, подсчитываться не будут. Это означает, что оценка не будет зависеть от операционной системы и установленных на ней программ, зависимость останется только от используемого процессора.

## 2.1. Подсчет количества тактов

Для того, чтобы подсчитать количество тактов процессора, необходимых для выполнения исследуемого алгоритма, предполагается выполнить действия, представленные на Рисунке 1.

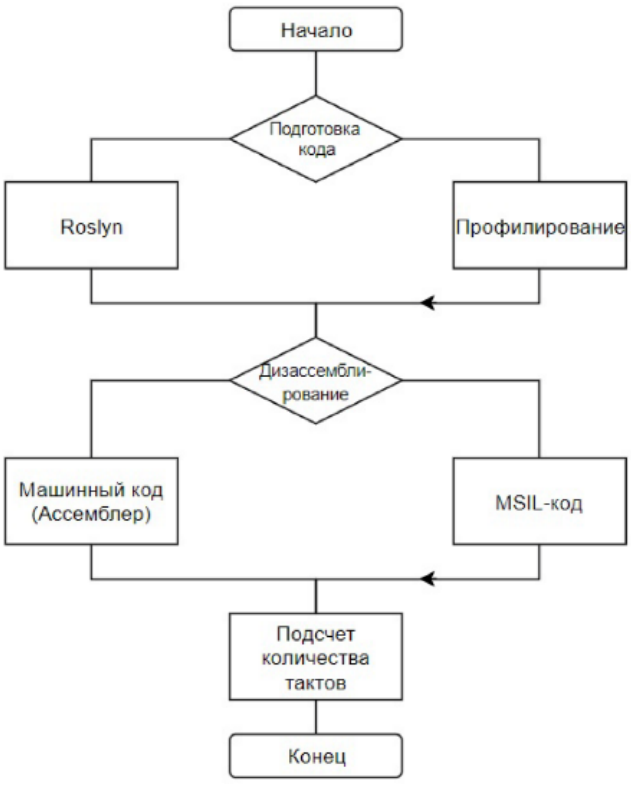


Рис. 1: Схема предлагаемого метода анализа алгоритмов.

На этапе подготовки кода к дизассемблированию было рассмотрено два подхода: использование Roslyn и использование профилировщиков и метрик кода.

Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, и т.д. Инструмент, используемый для анализа работы, называют профилировщиком [6].

Характеристики могут быть аппаратными (время) или вызванные программным обеспечением (функциональный запрос). Профилировщики помогают найти узкие места программы и в дальнейшем оптимизировать исходный алгоритм. После завершения процесса профилирования, разработчик может получить отчет по всем функциям, которые используются в программе. В отчет входит информация о вызывающей функции, о затраченном времени на исполнение выбранной функции и количество вызовов некоторых функций [7].

В рамках данной работы были рассмотрены два профилировщика: встроенный в Visual Studio и CodeAnalyst от AMD.

AMD CodeAnalyst Performance Analyzer для Windows — это набор мощных инструментов, которые помогают разработчикам оптимизировать производительность программного обеспечения. CodeAnalyst использует профилирование для идентификации и анализа точек доступа производительности в приложении, библиотеке, драйвере или модуле ядра. CodeAnalyst профилирует как управляемый, так и собственный код и собирает общесистемные данные профиля с низкими накладными расходами [8].

Несмотря на большой набор возможностей современных профилировщиков, с их помощью не удалось получить точную информацию по количеству циклов и ветвлениям, поэтому, следующим инструментом для изучения был выбран Roslyn.

### 2.1.1. Roslyn

Roslyn — платформа с открытым исходным кодом, разрабатываемая корпорацией Microsoft, и содержащая в себе компиляторы и средства для разбора и анализа кода, написанного на языках программирования C# и Visual Basic.

Roslyn используется в среде разработки Microsoft Visual Studio 2015. С помощью средств анализа, предоставляемых платформой Roslyn, можно производить полный разбор кода, анализируя все поддерживаемые конструкции языка.

Среда Visual Studio позволяет создавать на основе Roslyn как встраиваемые в саму IDE инструменты (расширения Visual Studio), так и независимые приложения (standalone инструменты).

Исходный код Roslyn доступен в соответствующем репозитории на GitHub [9]. Это позволяет посмотреть, какими функциями обладает данный инструмент и как они работают, а в случае обнаружения какой-либо ошибки — сообщить о ней разработчикам.

В Roslyn используются синтаксические деревья и семантические модели — сущности, на которых и базируется полноценный анализ. Синтаксическое дерево строится на основании исходного кода программы и используется для анализа различных конструкций языка. Семантическая модель предоставляет информацию об объектах и их типах.

Для получения синтаксического дерева (SyntaxTree) используется метод TryGetSyntaxTree или GetSyntaxTreeAsync экземпляра класса Document.

Семантическая модель (объект типа SemanticModel) получается из компиляции с использованием синтаксического дерева, полученного ранее. Для этого используется ме-

тод GetSemanticModel экземпляра класса Compilation, принимающего в качестве обязательного параметра объект типа SyntaxTree.

Класс, который будет обходить синтаксическое дерево и проводить анализ, должен быть унаследован от класса CSharpSyntaxWalker, что позволит переопределить методы обхода различных узлов. Вызывая метод Visit, принимающий в качестве параметра корень дерева (для его получения используется метод GetRoot объекта типа SyntaxTree), тем самым запускается рекурсивный обход узлов синтаксического дерева.

Для каждой конструкции языка определены узлы своего типа. А для каждого типа узла определён метод, выполняющих обход узлов подобного типа. Таким образом, добавляя обработчики (диагностические правила) в методы обхода тех или иных узлов, возможно анализировать только интересующие разработчиков конструкции языка.

Три основных элемента дерева — syntax nodes, syntax tokens, syntax trivia;

- Syntax nodes — синтаксические конструкции языка. К этой категории относятся объявления, определения, операторы и т.п.;
- Syntax tokens — лексемы, конечные символы грамматики языка. К этой категории относятся идентификаторы, ключевые слова, спец. символы и т.п.;
- Syntax trivia — дополнительная синтаксическая информация. К этой категории относятся комментарии, директивы препроцессора, пробелы и т.п.

Семантическая модель предоставляет семантическую информацию (об объектах, их типах и пр.), необходима для про-

ведения глубокого и сложного анализа. Для получения корректной семантической модели проект должен быть скомпилированным. Семантическую информацию об объекте предоставляет интерфейс ISymbol. Семантическую информацию о типе объекта предоставляет интерфейс ITypeSymbol. С помощью семантической модели возможно получение значений константных полей и литералов.

Можно выделить 3 наиболее часто используемые функции, предоставляемые семантической моделью:

- Получение информации об объекте;
- Получение информации о типе объекта;
- Получение константных значений.

Информацию об объекте предоставляют так называемые символы (symbols). Базовый интерфейс символа — ISymbol, предоставляет методы и свойства, общие для всех объектов, независимо от того, чем они являются: полем, свойством или чем-то ещё.

Существует ряд производных типов, выполняя приведение к которым можно получать более специфическую информацию об объекте. К таким интерфейсам относятся IFieldSymbol, IPropertySymbol, IMethodSymbol и прочие. Если использовать интерфейс IMethodSymbol, можно узнать, возвращает-ли метод какое-либо значение.

Для символов определено свойство Kind, возвращающее элементы перечисления SymbolKind. По своему предназначению это перечисление аналогично перечислению SyntaxKind. То есть с помощью свойства Kind можно узнать, с чем сейчас происходит работа: локальным объектом, полем, свойством, сборкой и пр. [10].

В ходе данной работы была написана программа на языке C# с использованием Roslyn, которая позволяет получить информацию о циклах в коде и преобразовать исходный код для последующего дизассемблирования. Исходный код программы доступен в репозитории на GitHub [11].

### 2.1.2. Дизассемблирование

Дизассемблирование — это способ получения исходного текста программы на языке ассемблера из программы, написанной на языке более высокого уровня. После получения кода на языке ассемблер можно переходить к подсчету количества тактов процессора, за которое выполняется алгоритм.

Чтобы получить машинный код для исходного алгоритма можно воспользоваться следующими средствами:

- стандартные средства Visual Studio;
- программа IDA Pro 6.6.

В среде разработки Visual Studio 2015, чтобы получить дизассемблированный код, необходимо открыть исходный код алгоритма и выполнить следующие действия:

1. поставить точку останова на одной из строчек программы,
2. запустить выполнение программы,
3. когда выполнение программы дойдет до точки останова, в меню необходимо выбрать “Окно” -> «Дизассемблирвоанный код».

Также была рассмотрена такая программа, как IDA Pro. IDA Pro — это интерактивный дизассемблер и отладчик одновременно [12]. Она позволяет превратить бинарный код про-

граммы в ассемблерный текст, который может быть применен для анализа работы программы.

Основная задача — превращение бинарного кода в читаемый текст программы. Ниже приведены некоторые возможности, уникальные для этой программы:

- развитая система навигации;
- система типов и параметров функций;
- встроенный язык программирования IDC;
- открытая и модульная архитектура;
- возможность работы с популярными процессорами;
- возможность работы с популярными форматами файлов.

В программе IDA Pro на вход принимается exe-файл программы. Однако, исходный код программ, написанных с использованием платформы .Net, на этапе сборки собирается в exe-файл, который представляет собой набор MSIL-инструкций. Таким образом, с помощью программы IDA Pro возможно просмотреть исходный код только на языке IL, который является более высокоуровневым по сравнению с языком ассмеблера, хоть и схожим, поэтому в дальнейшем, для анализа кода на языке ассемблер будет использоваться результат, полученный в ходе дизассемблирования с помощью Visual Studio.

### 2.1.3. Подсчет количества тактов

Для того, чтобы составить список подсчитываемых команд была использована официальная документация Intel, описанная в файле «Instruction tables» [5]. В данном документе описаны все команды процессора с учетом попадания и непопадания операции в регистры.

Регистры процессора — внутренние ячейки процессора, которые служат для хранения информации с практически мгновенным доступом. В отличии от оперативной памяти, для чтения и записи в регистры не нужно обращаться к внешнему устройству через шину, потому что регистры встроены в процессор и являются одной из его основных частей [13].

На Рисунке 2 представлены регистры процессора.

| Регистр | Старшие разряды | Имена 16-ти и 8-ми битных регистров | |
|---|---|---|---|
| | 31...16 | 15...8 | 7...0 |
| EAX | ... | AX | |
| | | AH | AL |
| EBX | ... | BX | |
| | | BH | BL |
| ECX | ... | CX | |
| | | CH | CL |
| EDX | ... | DX | |
| | | DH | DL |
| ESI | ... | SI | |
| EDI | ... | DI | |
| EBP | ... | BP | |
| ESP | ... | SP | |
| EIP | ... | IP | |

Рис. 2: Регистры процессора.

В данной работе используется процессор семейства Haswell — кодовое название микроархитектуры четвёртого поколения процессоров Intel Core.

В ходе выполнения данной работы была написана программа, которая подсчитывает количество команд на языке ассемблер и записывает их в файл. Исходный код програм-

мы доступен в репозитории на GitHub [11]. На вход программа принимает код, полученный после этапа дизассемблирования. Для определения названия ассемблер-команды и попадания или непопадания операции в регистры используются регулярные выражения и функции работы со строками [14]. Для каждой обнаруженной ассемблер-команды счетчик количества тактов увеличивается на соответствующее ей значение тактов, взятое из документа «Instruction tables» [5].

Следует отметить, что до учета попадания операндов в регистры — результат был прямо противоположный.

Для оценки работоспособности написанной программы были рассмотрены два алгоритма вычисления факториала, реализованные на языке C#. Первый алгоритм вычисляет значение факторила по определению, второй алгоритм – с помощью факторизации. Исходный код алгоритмов вычисления факториалов представлен в Приложении.

На Рисунке 3 представлены результаты сравнения алгоритмов по времени при $n = 50000$.

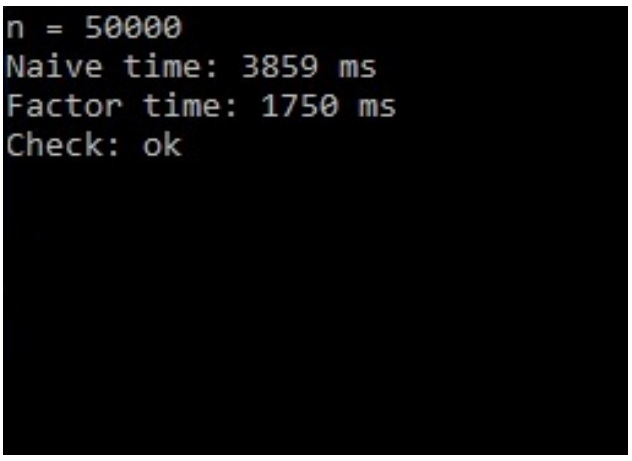


Рис. 3: Результаты сравнения по времени.

На Рисунке 4 представлены результаты сравнения алгоритмов по количеству тактов процессора при $n = 50000$.

Из полученных результатов видно, что алгоритм вычис-

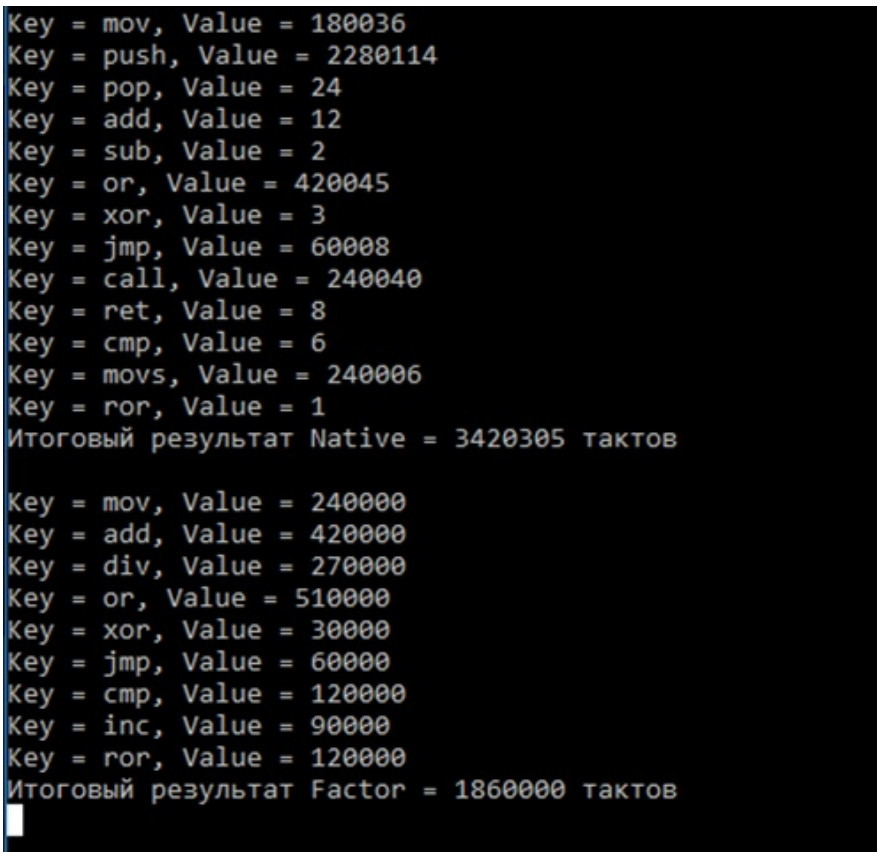


Рис. 4: Результаты сравнения по количеству тактов процессора.

ления факториала по определению работает почти в два раза медленнее по времени и выполняется за почти в 2 раза большее количество операций. Из количества подсчитанных команд видно, что это связано с большим количеством обращения к памяти в случае вычисления факториала по определению.

Несмотря на то, что результат был получен довольно быстро, такой подход не представляется возможным автоматизировать и расширить для использования в практически-значимых задачах. Roslyn оказался удобен для примитивных программ и позволил получить информацию о циклах и ветвлениях, но в дальнейшем выбранный инструмент не сможет покрыть все возможные языковые конструкции, а лишь небольшую часть частных случаев. Поэтому на следующем этапе работы был рассмотрен другой способ получения значений для функций трудоемкости.

## 2.2. Подсчет количества инструкций

Задача получения количества тактов процессора оказалась достаточно трудоемкой, поэтому далее был рассмотрен следующий инструмент: Intel VTune Amplifier, который можно использовать для подсчета количества инструкций процессора. Как упоминалось выше, под инструкциями понимаются микрооперации (uops), на которые разбиваются макроинструкции Intel, такие как ADD, MOV, SUB и т. д.

Intel VTune Amplifier — это инструмент анализа производительности, предназначенный для пользователей, разрабатывающих последовательные и многопоточные приложения. Данный инструмент позволяет производить анализ на платформах Windows, Linux и macOS.

В системах Windows VTune Amplifier интегрируется в программное обеспечение Microsoft Visual Studio, а также доступен как автономный клиент GUI. В системах Linux и macOS VTune Amplifier работает исключительно как отдельный клиент GUI. Также, во всех поддерживаемых системах можно использовать интерфейс командной строки для сбора данных и построения отчетов [15].

Intel VTune Amplifier представляет следующие основные типы анализа производительности:

- Предопределенные типы анализа:
  - Анализ алгоритмов
  - Анализ вычислений с интенсивным использованием
  - Анализ микроархитектуры
  - Анализ платформы
- Типы пользовательского анализа

Каждая категория представляет собой ветвь в дереве анализа производительности. Можно выбрать любой из доступных типов анализа либо из графического интерфейса продукта, либо с помощью интерфейса командной строки.

Анализ алгоритмов включает следующие типы анализа:

- Basic Hotspots: идентифицируются функции, которые занимают наибольшее время процессора для выполнения и восстанавливается дерево вызовов для каждой функции.
- Advanced Hotspots: контролируется все программное обеспечение, выполняемое в системе, включая модули операционной системы.
- Concurrency: определяются функции, которые занимают наибольшее время процессора для выполнения, и показывается, насколько хорошо приложение оптимизировано для существующего количества логических ядер процессора.
- Locks and Waits: идентифицируются объекты, которые могут вызвать неэффективное использование ЦП.
- Memory Consumption: анализируется потребление оперативной памяти и определяются выделенные и освобожденные во время анализа объекты памяти.

Анализ вычислений с интенсивным использованием применяется для общего анализа производительности приложений и для понимания производительности приложения в целом, чтобы впоследствии перейти к более узконаправленным типам анализа. Интенсивный анализ приложений включает в себя характеристику производительности HPC. Данная характеристика оценивает вычислительную и пропускную способность приложения для работы с данными с плавающей точкой и анализирует эффективность использования памяти.

В рамках анализа микроархитектуры представлены типы анализа, которые могут быть использованы для оценки эффективности использования кода на современном оборудовании.

В разделе «Анализ платформы» представлены типы анализов, отслеживающие использование процессора, графического процессора и мощности для исследуемого приложения или системы.

Все типы анализа, предоставляемые VTune Amplifier в окне «Тип анализа» предопределены разработчиками Intel и содержат набор параметров по умолчанию. Также у пользователя есть возможность создавать собственные типы анализа на основе существующих предопределенных конфигураций [16].

В данной работе используется анализ вычислений с интенсивным использованием. Пример использования данного типа анализа приведен на Рисунке 5.

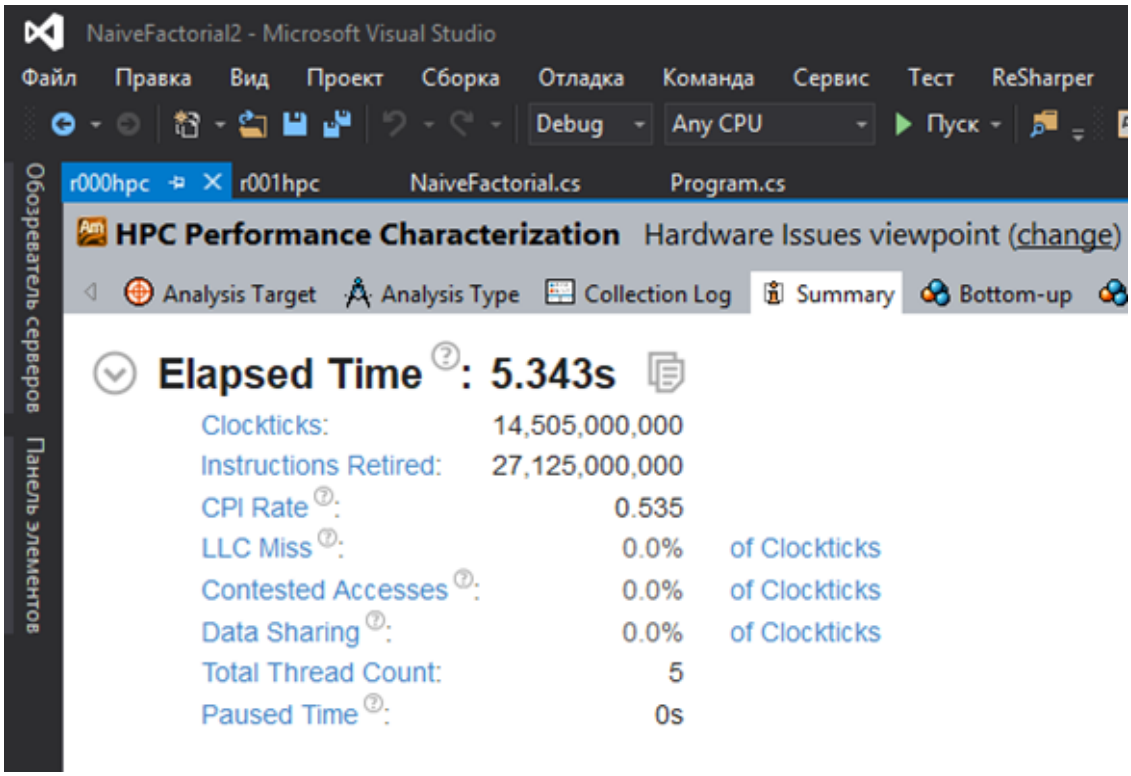


Рис. 5: Пример использования VTune Amplifier

В качестве количественной меры для трудоемкости выбран

параметр Instructions Retired. Instructions Retired — это числовая характеристика производительности оборудования, которая показывает, сколько команд было выполнено, т. е. инструкции, которые действительно необходимы потоку выполнения программы.

Современные процессоры выполняют гораздо больше инструкций, чем действительно необходимо для потока программы. Такое поведение связано с взаимодействием с процессором операционной системы и другого окружения и с осуществлением переходов в процессе обработки ветвлений в исследуемом алгоритме. Для реализации переходов в современных процессорах используются предсказатели переходов, которые хранят историю переходов и предсказывают на основании имеющихся данных следующий переход, который не всегда является верным.

Счетчик Instructions Retired включает в себя только те инструкции и команды, которые необходимы для выполнения исследуемого алгоритма и соответствуют правильным переходам. Инструкции и ошибки неверно предсказанных переходов удаляются при идентификации неверного предсказания, и в итоге учитываются только правильные пути [17].

Таким образом, при подсчете параметра «Instruction retried» VTune Amplifier дает гораздо более объективный счет благодаря тому, что он не учитывает выполнение «ненужных» инструкций и такой способ получения количественной меры подходит для определения трудоемкости.

# 3. Количественная мера информационной чувствительности

Для получения количественной меры информационной чувствительности в данной работе используется статистическая оценка, определяемая по формуле (6) и описанная в Главе 1.

Для дальнейшего анализа был выбран пакет Concorde, позволяющий получить решение задачи коммивояжера.

Задача коммивояжера (TSP) — одна из наиболее интенсивно изучаемых проблем вычислительной математики. Задача заключается в поиске кратчайшего маршрута, проходящего через указанные точки хотя бы по одному разу с последующим возвратом к исходной точке [18].

В качестве исходных данных был выбран пакет для решения задачи коммивояжера — Concorde.

Concorde — это библиотека для решения задачи коммивояжера (TSP) и других проблем оптимизации сети, связанных с этой задачей. Код написан на языке программирования ANSI C и доступен для использования в научных исследованиях.

Concorde был использован для получения оптимальных решений для всех 110 экземпляров TSPLIB; самый крупный из которых составляет 85 900 городов.

Библиотека Concorde включает в себя более 700 функций, позволяющих пользователям создавать специализированные коды для TSP-подобных задач. Все функции Concorde являются потокобезопасными для программирования в параллельных средах с разделяемой памятью. Также, Concorde включает в себя код для работы в системе UNIX и поддерживает возможность решения задач линейного программирования с использованием QSopt [19].

QSopt — это библиотека, которая содержит функции для

использования в таких приложениях, как задача коммивояжера или смешанное целочисленное программирование. QSopt также может использоваться для решения широкомасштабных задач линейного программирования [20].

Исследования проводились с использованием ресурсного центра Научного парка СПбГУ «Вычислительный центр».

## 3.1. Генерация исходных данных

В данной работе исследуются два типа графов, представленных на Рисунках 6 и 7.

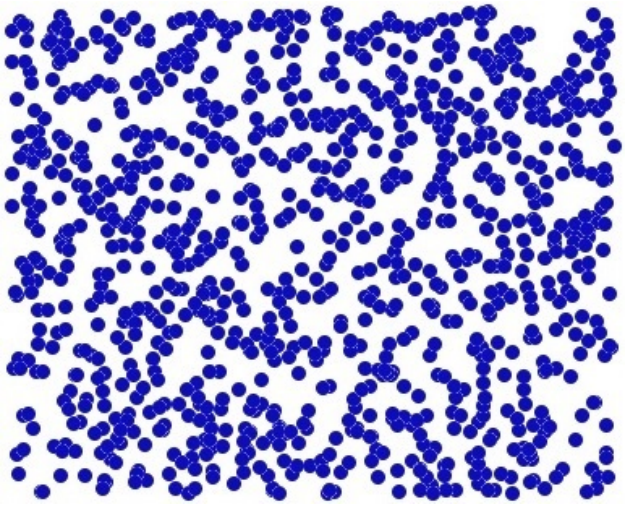

Рис. 6: Граф с равномерно распределенными на плоскости вершинами

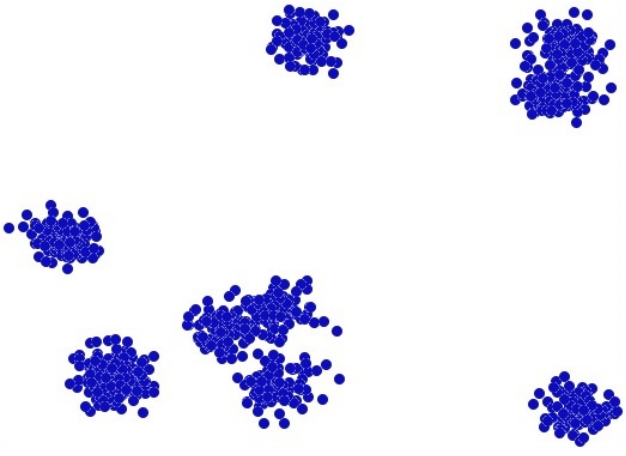

Рис. 7: Граф с равномерно распределенными на плоскости кластерами вершин

Для генерации графов с равномерно распределенными на плоскости вершинами используется генератор portgen. Вершины графа — точки на плоскости с целочисленными координатами, каждая из которых выбирается из интервала $[0, 10^6)$. Веса ребер соответствуют евклидовым расстояниям, округленным до ближайшего целого.

Для генерации графов с равномерно распределенными на плоскости вершинами используется генератор portcgen. Выбирается $N/100$ центров кластеров с равномерно распределенными целочисленными координатами из интервала $[0, 10^6)$. Для каждой из $N$ вершин случайно выбирается центр и две нормально распределенные величины, которые умножаются на $10^6/\sqrt{N}$ и после округления добавляются к соответствующим координатам выбранного центра. Веса ребер также являются евклидовыми расстояниями, округленными до ближайшего целого.

Для того, чтобы оценить возможность применения предложенной методики, необходимо вычислять информационную чувствительность в зависимости от времени выполнения и от количества базовых операций, предложенных в данной работе. Такой подход необходим для сравнения результатов, полученных с помощью уже существующих методик, и результатов, полученных с помощью предлагаемой методики.

Для сбора необходимого количества данных необходимо многократно запускать программную реализацию алгоритма на случайных входных данных выбранного типа. Для этого был написан скрипт, который выполняет следующие действия:

1. генерация исходного графа (использование portgen или portcgen)

2. запуск анализа vtune, который включает в себя подсчет ко-

личества базовых инструкций

3. построение отчета по полученным vtune данным
4. использование time для вычисления тактового времени выполнения алгоритма
5. запись полученных результатов в файл
6. удаление временных файлов

Исходный код реализованных скриптов представлен в репозитории на GitHub [11].

## 3.2. Построение количественной меры информационной чувствительности

С помощью скриптов, описанных ранее, были получены значения функций трудоемкости по времени и по количеству базовых операций.

Для практических расчетов информационной чувствительности необходимо знать объем выборки. Для $n > 50$ и $\gamma = 0,95$, где $n$ - размер выборки, а $\gamma$ - доверительная вероятность, применима формула:

$$n^* = \frac{3,8416 V_f^2}{\varepsilon^2} \tag{8}$$

Сначала значение $V_f$ оценивается на основе предварительного эксперимента, т. е. выборки заранее определенного объема $n$, например, $n = 50$. Затем по формуле (8) рассчитывается значение $n^*_{(1)}$ экспериментов с программной реализацией алгоритма, определяется новое выборочное значение $V_f$ и рассчитывается $n^*_{(2)}$. Если полученный объем выборки $n^*_{(2)} < n^*_{(1)}$, то эксперимент останавливается, иначе выполняется очередная итерация до выполнения указанного условия. При выполнении

условия останова $n^*_{(2)} < n^*_{(1)}$ значение $\varepsilon$ может быть скорректировано на основе нового коэффициента вариации и предыдущего (т. е. большего) значения объема выборки [3].

Для выборки найденного размера необходимо вычислить следующие величины:

- выборочное среднее $\overline{f_v} = \frac{1}{n}\sum_{i=1}^{n} f_i$
- выборочная дисперсия $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(f_i - \overline{f_v})^2$
- стандартное отклонение $\sigma = \sqrt{s^2}$
- выборочный коэффициент вариации $V_f = \frac{\sigma}{\overline{f_v}}$
- минимальное и максимальное значение функции трудоемкости
- нормированный размах варьирования $R_N(n) = \frac{f^{\wedge}_A(n) - f^{\vee}_A(n)}{f^{\wedge}_A(n) + f^{\vee}_A(n)}$

### 3.2.1. Расчеты по количеству базовых операций

Промежуточные значения и итоговые результаты расчетов количественной меры информационной чувствительности по количеству базовых операций для задач с равномерно распределенными на плоскости вершинами представлены в Таблицах 1, 2.

Таблица 1. Характеристики выборки для задач с равномерно распределенными на плоскости вершинами

| Размер | $n^*$ | $\overline{f_v}$, млрд | $\sigma$, млрд | $min$, млрд | $max$, млрд |
|---|---|---|---|---|---|
| 100 | 816 | 3,255 | 4,191 | 0,768 | 13,464 |
| 250 | 137 | 11,859 | 7,494 | 4,185 | 26,526 |
| 350 | 328 | 23,462 | 18,274 | 6,036 | 58,101 |
| 500 | 414 | 42,327 | 26,531 | 8,676 | 89,706 |

Для вычисления статистической количественной меры информационной чувствительности используется формула (6) из Главы 1.

Таблица 2. Вычисление значения информационной чувствительности

| Размер | $V_f$ | $R_N(n)$ | $\delta_{IS}$ |
|---|---|---|---|
| 100 | 1,28755 | 0,89207 | 1,14859 |
| 250 | 0,63189 | 0,72746 | 0,45968 |
| 350 | 0,74803 | 0,81178 | 0,60724 |
| 500 | 0,62682 | 0,82363 | 0,51626 |

Промежуточные значения и итоговые результаты расчетов количественной меры информационной чувствительности по количеству базовых операций для задач с равномерно распределенными на плоскости кластерами вершин представлены в Таблицах 3, 4.

Таблица 3. Характеристики выборки для задач с равномерно распределенными на плоскости кластерами вершин

| Размер | $n^*$ | $\overline{f_v}$, млрд | $\sigma$, млрд | $min$, млрд | $max$, млрд |
|---|---|---|---|---|---|
| 100 | 200 | 1,995 | 0,835 | 1,269 | 3,762 |
| 250 | 165 | 13,925 | 11,125 | 4,338 | 39,288 |
| 350 | 564 | 31,250 | 17,607 | 10,602 | 50,160 |
| 500 | 747 | 61,540 | 53,010 | 21,831 | 144,732 |

Таблица 4. Вычисление значения информационной чувствительности

| Размер | $V_f$ | $R_N(n)$ | $\delta_{IS}$ |
|---|---|---|---|
| 100 | 0,41891 | 0,49553 | 0,20758 |
| 250 | 0,79891 | 0,80113 | 0,64003 |
| 350 | 0,56342 | 0,65103 | 0,36680 |
| 500 | 0,86140 | 0,73786 | 0,63560 |

### 3.2.2. Расчеты по времени

Промежуточные значения и итоговые результаты расчетов количественной меры информационной чувствительности по времени для задач с равномерно распределенными на плоскости вершинами представлены в Таблицах 5, 6.

Таблица 5. Характеристики выборки для задач с равномерно распределенными на плоскости вершинами

| Размер | $n^*$ | $\overline{f_v}$ | $\sigma$ | $min$ | $max$ |
|---|---|---|---|---|---|
| 100 | 593 | 1,00125 | 1,73518 | 0,19 | 4,28 |
| 250 | 188 | 4,27125 | 3,49233 | 1,34 | 11,37 |
| 350 | 437 | 7,37750 | 6,61059 | 1,62 | 18,18 |
| 500 | 876 | 13,50750 | 9,60146 | 2,25 | 22,61 |

Таблица 6. Вычисление значения информационной чувствительности

| Размер | $V_f$ | $R_N(n)$ | $\delta_{IS}$ |
|---|---|---|---|
| 100 | 1,73302 | 0,91499 | 1,58569 |
| 250 | 0,81764 | 0,78914 | 0,64523 |
| 350 | 0,89605 | 0,87230 | 0,78162 |
| 500 | 0,71082 | 0,81899 | 0,58216 |

Промежуточные значения и итоговые результаты расчетов количественной меры информационной чувствительности по времени для задач с равномерно распределенными на плоскости кластерами вершин представлены в Таблицах 7, 8.

Таблица 7. Характеристики выборки для задач с равномерно распределенными на плоскости кластерами вершин

| Размер | $n^*$ | $\overline{f_v}$ | $\sigma$ | $min$ | $max$ |
|---|---|---|---|---|---|
| 100 | 202 | 0,72500 | 0,74123 | 0,35 | 1,79 |
| 250 | 472 | 3,07125 | 1,95077 | 1,08 | 5,97 |
| 350 | 261 | 11,33500 | 11,94466 | 3,29 | 35,58 |
| 500 | 278 | 22,78875 | 27,00901 | 5,90 | 77,32 |

Таблица 8. Вычисление значения информационной чувствительности

| Размер | $V_f$ | $R_N(n)$ | $\delta_{IS}$ |
|---|---|---|---|
| 100 | 0,63517 | 0,69362 | 0,44057 |
| 250 | 1,18519 | 0,85821 | 1,01714 |
| 350 | 1,02239 | 0,67290 | 0,68796 |
| 500 | 1,05379 | 0,83072 | 0,87540 |

Т. к. $\frac{1}{\sqrt{3}} \leq \delta_{IS}(n) \leq 1$, значит рассмотренный в данной работе алгоритм относится к сильно чувствительным к входным данным (классификация алгоритмов описана в [3]).

Более подробный анализ полученных в данной главе результатов будет представлен в следующем разделе.

## Тестирование

Для удобства сравнения результатов, полученных с помощью различных методик, были построены графики зависимости значений информационной чувствительности от размера входных данных.

На Рисунке 8 представлены результаты сравнения значений информационной чувствительности, вычисленной с помощью количества базовых операций. График, подписанный как portgen, соответствует задачам с равномерно распределенными на плоскости вершинами, portcgen - задачам с равномерно распределенными на плоскости кластерами вершин.

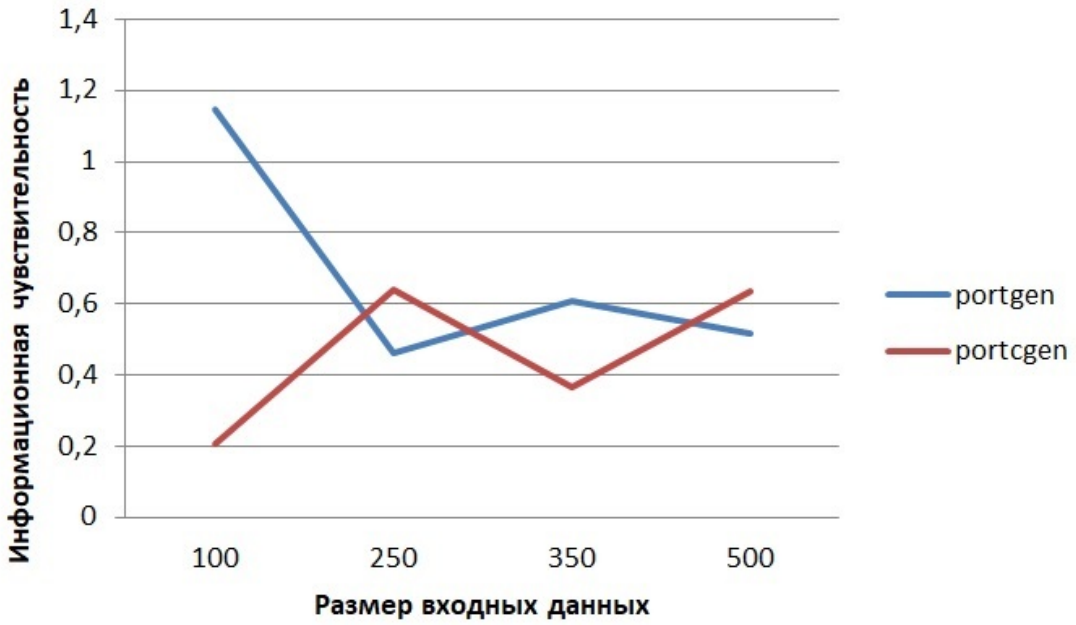


Рис. 8: Результаты сравнения по количеству операций.

На Рисунке 9 представлены результаты сравнения значений информационной чувствительности, вычисленной с помощью временной оценки.

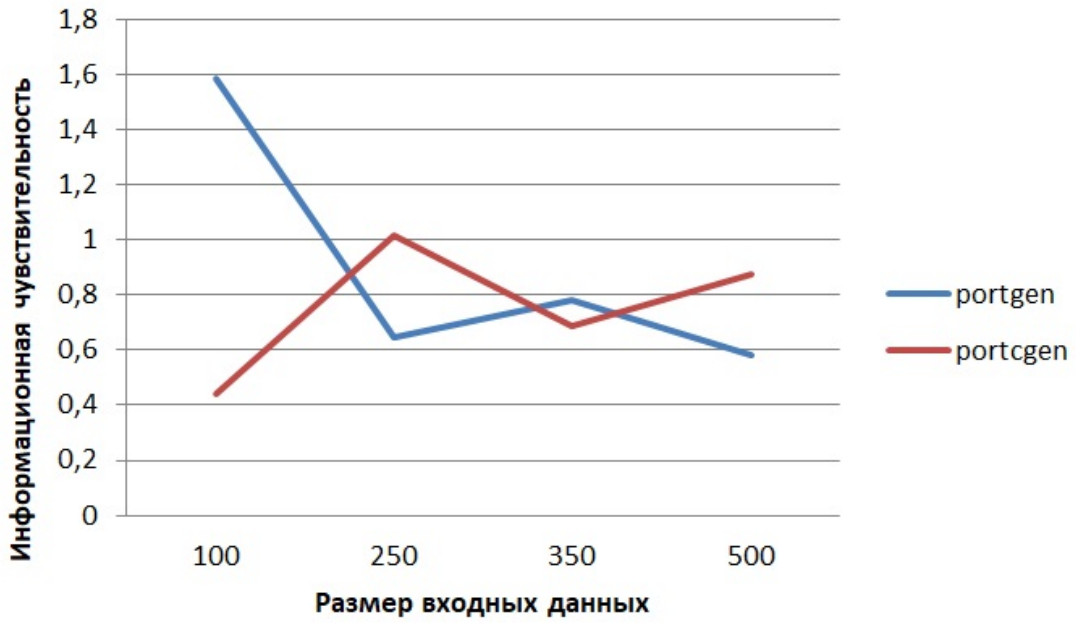


Рис. 9: Результаты сравнения по времени.

На представленных рисунках видно, что форма и вид графиков отличаются не существенно. Также, следует отметить, что значения информационной чувствительности, полученные с помощью подсчета количества базовых операций, меньше, чем значения, полученные с помощью использования временной оценки. Такой результат означает, что разработанная в ходе данной работы методика не противоречит уже существующим методам анализа компьютерных алгоритмов и дает более объективную оценку эффективности компьютерных алгоритмов. Таким образом, предложенная методика может быть применима для анализа компьютерных алгоритмов.

## Заключение

В ходе выполнения данной работы были исследованы существующие методы анализа алгоритмов, основанные на временной оценке. Были рассмотрены два способа получения значений функции трудоемкости по количеству базовых операций и выбран один из них, подходящий для дальнейшего использования. По полученным значениям функций трудоемкости были вычислены значения информационной чувствительности и обоснована возможность применения предложенной методологии.

Таким образом, был разработан и применен метод анализа информационной чувствительности компьютерных алгоритмов, который дает более достоверную оценку при прогнозировании эффективности компьютерного алгоритма по сравнению с временной оценкой. Оценка трудоемкости программной реализации алгоритма может проводиться независимо от окружения ПК, однако сохраняется зависимость от архитектуры процессора.

## Список используемой литературы


1. Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы, 2008. 303 с.

2. A RISC Computer for the Third Millennium Knuth's official MIX page

   https://www-cs-faculty.stanford.edu/ knuth/mmix.html

3. Петрушин М. В., Ульянов М. В. Информационная чувствительность компьютерных алгоритмов, 2010. 224 с.

4. Количественные оценки информационной чувствительности алгоритмов

   http://www.ipu.ru/sites/default/files/publications/31334/13582-31334.pdf

5. Instruction tables

   http://www.agner.org/optimize/instruction_tables.pdf

6. Профилирование

   http://studbooks.net/2238122/informatika/profilirovanie

7. Касперски К. Техника оптимизации программ. Эффективное использование памяти. BHV, 2003. 560 с.

8. CodeAnalyst для Windows https://developer.amd.com/tools-and-sdks/archive/codeanalyst-performance-analyzer-for-windows/

9. The .NET Compiler Platform ("Roslyn") provides open-source C# and Visual Basic compilers with rich code analysis APIs. https://github.com/dotnet/roslyn

10. Введение в Roslyn. Использование для разработки инструментов статического анализа. https://www.viva64.com/ru/b/0399/ID0ERLAC

11. https://github.com/quakke/metody_analiza

12. IDA Pro – интерактивный дизассемблер

    http://www.idasoft.ru/idapro/

13. Регистры процессора

    http://cpubook.ru/Регистры%20Процессора

14. Рихтер Дж. CLR via C#, Third edition. Питер, 2012. 928 с.

15. Vtune Amplifier Introduction https://software.intel.com/en-us/vtune-amplifier-help-introduction

16. Analyze Performance https://software.intel.com/en-us/vtune-amplifier-help-analyze-performance

17. Instructions Retired Event https://software.intel.com/en-us/vtune-amplifier-help-instructions-retired-event

18. Алгоритмы для задачи коммивояжёра

    https://www.lektorium.tv/lecture/14235

19. Concorde TSP Solver

    http://www.math.uwaterloo.ca/tsp/concorde.html

20. QSopt Linear Programming Solver

    https://www.math.uwaterloo.ca/ bico/qsopt/

# Приложение 1

Исходный код алгоритма для вычисления значения факториала по определению на языке C#:

```
static BigInteger FactNaive(int n) {
    BigInteger r = 1;
    for (int i = 2; i <= n; ++i)
        r *= i;
    return r;
}
```

Исходный код алгоритма для вычисления значения факториала с помощью факторизации на языке C#:

```
static BigInteger FactFactor(int n) {
    if (n < 0) return 0;
    if (n == 0) return 1;
    if (n == 1 || n == 2) return n;
    bool[] u = new bool[n + 1];
    List<Tuple<int, int>> p = new List<Tuple<int, int>>();
    for (int i = 2; i <= n; ++i)
        if (!u[i])
        {
            int k = n / i;
            int c = 0;
            while (k > 0) {
                c += k;
                k /= i;
            }
            p.Add(new Tuple<int, int>(i, c));
            int j = 2;
            while (i * j <= n) {
                u[i * j] = true;
                ++j;
            }
    }
    BigInteger r = 1;
    for (int i = p.Count() - 1; i >= 0; --i)
        r *= BigInteger.Pow(p[i].Item1, p[i].Item2);
    return r;
}
```