МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Южно-Уральский государственный университет (национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук

Кафедра вычислительной математики и высокопроизводительных вычислений

РАБОТА ПРОВЕРЕНА
Рецензент,
_____ /ФИО
« » 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, к.ф.-м.н., доцент
_____ /Н.М. Япарова
« » 2020 г.

«Алгоритм решения задачи линейного программирования в условиях неполных данных»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ-090401.2020.683 ПЗ ВКР

Руководитель работы, к. ф.-м. н., доцент
_____ /И.М. Соколинская
« » 2020 г.

Автор работы
Студент группы КЭ-230
_____ /И.Ю. Никоноров
« » 2020 г.

Нормоконтролер, к. ф.-м. н., доцент
_____ /С.У. Турлакова
« » 2020 г.

Челябинск 2020

АННОТАЦИЯ

Никоноров И.Ю. Алгоритм решения задачи линейного программирования в условиях неполных данных. - ЮУрГУ, ВШЭКН, ФГАОУ ВО ЮУРГУ (НИУ); 2020, 66 с., 23 ил., библиогр. список - 36 наим.

Данная выпускная квалификационная работа посвящена алгоритму решения задачи линейного программирования в условиях неполных данных. Под неполными данными понимается наличие в задаче линейного программирования некоторого неформализованного ограничения, влияющего на общее решение задачи. Для решения подобных задач использовать стандартные методы линейного программирования не представляется возможным. Предлагаемый в данной работе алгоритм для решения задач такого рода является сплавом методов линейного программирования и дискриминантного анализа. Блок дискриминантного анализа предназначен для выявления вида неформализованного ограничения. Блок линейного программирования используется для решения аппроксимационной задачи линейного программирования - исходной задачи, в которую добавлено найденное формализованное ограничение.

Помимо анализа предметной области в рамках выполнения выпускной квалификационной работы данный алгоритм был реализован в виде клиентсерверного приложения на стеке технологий Vue.js (реализация клиентской части) и Node.js (реализация серверной части). Работа приложения была протестирована на задачах линейного программирования размерностью $n < 100$. Также был проведен ряд вычислительных экспериментов на предмет исследования влияния параметров алгоритма на его быстродействие.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В практике экономико-математического моделирования часто возникают задачи линейного программирования с ограничениями, не поддающимися полной формализации.

Как правило задачи такого типа появляются в технологическом процессе, параметры которого невозможно измерить напрямую в силу каких-либо обстоятельств.

Единственным критерием, определяющим качество значений параметров в данном случае, может быть решение некоторого эксперта, основанное на каких- либо эмпирических данных, в качестве которого может выступать человек, нейронная сеть или экспертная система.

Для решения задач подобного рода может использоваться универсальный алгоритм, представляющий собой некоторый сплав методов из различных областей математического программирования. Одним из таких алгоритмов является алгоритм решения задачи линейного программирования в условиях неполных данных.

Поэтому, целью данной работы является обзор и реализация алгоритма решения задачи линейного программирования в условиях неполных данных. Для достижения данной цели необходимо решить следующие задачи:

1) изучить методы решения задач линейного программирования;
2) изучить методы решения задач с неформализованными ограничениями;
3) изучить общие принципы и методы теории распознавания образов;
4) изучить методы решения задач дискриминантного анализа.

Задачи линейного программирования с неформализованными ограничениями имеют непосредственное отношение к задачам различных сфер жизни общества. Это могут быть, например, задачи из сферы черной металлургии, задачи о портфеле ценных бумаг и т.п. Качественное использование алгоритма решения задачи линейного программирования в условиях неполных данных позволит улучшить не

только производственную способность и функциональные возможности технологического процесса, но и как следствие жизнь каждого из нас.

1 ТЕОРИЯ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

1.1 Основные понятия

Общей задачей линейного программирования (ЗЛП) называется задача, состоящая в поиске массива действительных чисел $X = (x_1, x_2, ..., x_n)$, при которых целевая функция вида

$$L(X) = C_1X_1 + C''X'' + - + c\#\, x\# \tag{1}$$

достигает экстремума и удовлетворяет системе ограничений

$$a_{ц}X_1 + a\$''X_2 + - + a_{ы}\, x\# = b\$\ (i = 1, m) \tag{2}$$

$$a\$_1x_1 + a_{i2}x_2 + —+ a_{in}x_n < b\$, i = m + 1, m + s \tag{3}$$

$$\begin{aligned} \text{^\$1^1} + 0\text{-}i2\%2 + ¯ + O\text{-}inX\# — \ , \ i \\ = m + s + 1, \ m + s + p \end{aligned} \tag{4}$$

$$x\% — 0, j = 1, \ n \tag{5}$$

Любой n-мерный вектор $X = (x_1, x_2, \ldots, x_n)$, удовлетворяющий системе ограничений (2)-(4) и условиям неотрицательности (5) называется допустимым решением (планом) ЗЛП. Все допустимые решения задачи формируют область допустимых решений (ОДР), а то решение, при котором целевая функция достигает экстремума, называется оптимальным.

Все ЗЛП по виду целевой функции и ограничений делятся на два типа:

1) Каноническая ЗЛП Она

записывается в виде:

$$L(X) = \wedge^{n} c\%x\% \wedge max(min)$$

$$\wedge_{J=1}^{n} a\$j\ XJ = b\$\ ,i = l,r \qquad (6)$$

$$Xj > 0,y = l\ ,\ n$$

2) Симметрическая ЗЛП

Она записывается в виде:

$$L(X) = \wedge_{J=1}^{n} c\%\ x\% \wedge \max x(min)$$

$$\wedge_{J=1}^{n} a_{t}jx\% < b\$,\ i = l,\ r \qquad (7)$$

$$x\% > 0,j = l,\ n$$

или

$$L\{X) = \wedge_{J=1}^{n} cjXJ \wedge \min$$

$$: a_{t}jXj > b\$,i = l,r \qquad (8)$$

$$Xj > 0,j = l,\ n$$

1.2 Методы линейного программирования

Для решения ЗЛП используются несколько методов: 1) графический;

2) симплекс-метод.

1.2.1 Графический способ Рассмотрим

ЗЛП с двумя переменными:

$$L(X) = c_1x_1 + c_2x_2 \wedge extr$$

$$\begin{cases} a_{\varepsilon 1}\, x_{\pm} + a_{\pm 2}x_2 < (>)Ь_1 \\ {}^{a}21^{x}1 + {}^{a}22^{x}2 < (>)\wedge 2 \\ От1*1 + \&m2^{x}2 < (>)\wedge ш\, x_1 \\ > 0, x_2 > 0 \end{cases}$$

Подобная задача может быть решена графически в силу того, что ОДР здесь представляет собой многоугольник, стороны которого лежат на прямых, получаемых из системы ограничений задачи и вершинами, доставляющими экстремум целевой функции.

Алгоритм графического решения ЗЛП:

1) Построить ОДР;

2) Построить градиент целевой функции;

3) Построить опорную прямую, перпендикулярную градиенту целевой функции;

4) Перемещая опорную прямую в направлении градиента целевой функции, определить «точку входа» и «точку выхода» (первая и последняя встретившаяся опорной прямой точки из ОДР соответственно);

5) Определить координаты оптимальной точки (точки входа или точки выхода) и вычислить значение целевой функции в ней.

1.2.2 Симплекс-метод

Симплекс-метод состоит из ряда шагов:

1) Поиск начального базисного решения

Существует два основных способа поиска начального базисного решения:
- Преобразования Жордана-Гаусса [23, с. 9]
- М-задача

Задача (6) записывается в расширенной форме, когда в каждое из уравнений задачи, записывается по одной новой переменной, которые называются искусственными:

$$L(X) = \sum_{j=1}^{n} c_j x_j - M \sum_{i=1}^{r} x_{n+i} \to \max(min)$$

$$\begin{cases} a_{ij} x_j + x_{n+i} = b_i, i = \overline{1,r} \\ x_j \geq 0, j = \overline{1,n} \\ x_{n+i} \geq 0, i = \overline{1,r} \end{cases} \quad (9)$$

В этом случае переменные x_{n+} являются базисными и начальное базисное решение имеет вид:

$$X_{n+i}, i = \overline{1,m}$$
$$x_j = 0, x_n = 0$$

2) Переход от одного базисного решения к другому

Полученное базисное решение проверяют на оптимальность - если оно не оптимально, то одну переменную из базиса заменяют на другую, получая тем

самым новый базис. Так делается до тех пор, пока получившийся базис не будет являться оптимальным.

Пошаговый алгоритм перехода от одного базиса к другому представлен в [27, с. 319-323].

2 ТЕОРИЯ ДИСКРИМИНАНТНОГО АНАЛИЗА

Дискриминантный анализ - совокупность методов, используемых в теории распознавания образов с целью нахождения комбинации признаков, которая характеризует или разделяет два или более классов.

2.1 Основные понятия

Если имеется два конечных множества $X_г$, содержащего объекты первого класса и X_2, содержащего объекты второго класса, то задача дискриминантного анализа формулируется следующим образом: требуется найти некоторую функцию /, удовлетворяющую системе неравенств

$$\begin{gathered} f(x) > 0 \ (yx \ e \ X_г) \\ /O) < 0 \ (V^* \ e \ X'') \\ f \ e \ F \end{gathered} \qquad (10)$$

где F - класс функций /.

Функция /, решающая систему (10) называется разделяющей или дискриминантной [8, с. 132].

2.2 Методы дискриминантного анализа

Методов дискриминантного анализа достаточно много и нет никакого смысла описывать каждый из них, поэтому к рассмотрению предлагается классификация методов дискриминантного анализа по Еремину и Мазурову, описанная ими в [8].

Рассмотрим задачу дискриминантного анализа для двух множеств: найти элемент $x^* \ e \ X^* \ c \ \{R^n \wedge R\}$ такой, чтобы выполнялись неравенства при $X_г \ c \ R^n$, X_2 с R^n:

$$x^*(x) > 0 \ (\forall x \in X_г) \ x$$
$$^*(x) < 0 \ (\forall x \in X_2) \qquad (11)$$

Если X^* - класс аффинных функционалов, т.е. $x^*(x) = (y^*, x) + z^*$, где $y^* \in R^n$, $z^* \in R$, то задача нахождения хотя бы одного решения системы линейных неравенств (11) называется задачей аффинного дискриминантного анализа [8, с. 138].

Использование аффинных функционалов в качестве разделяющих оправдывается их простой структурой и технической реализуемостью, а также тем, что линейный случай дает большие возможности для математического анализа и строгого обоснования вычислительных процедур. Кроме того, многие нелинейные методы дискриминантного анализа в качестве основного блока содержат модуль разделения конечных множеств с помощью аффинного функционала (например, метод комитетов, метод потенциалов, и др.) [8, с. 138].

Рассматриваемые ниже методы относятся к числу непараметрических, т.е. считается неизвестным какое бы то ни было статистическое описание образцов. Поэтому за критерий качества классификации, осуществляемой с помощью найденной разделяющей функции, принимается число безошибочных классификаций элементов множества $X_г$ и X_2 [8, с. 138].

Решение задачи аффинного дискриминантного анализа в принципе может быть осуществлено с помощью любого метода решения система линейных неравенств. Однако задачи распознавания образов обладают определенной спецификой, которая влечет свои требования к организации вычислительных процедур. Эта специфика состоит в том, что разделяемые множества $X_г$ и X_2, элементы которых формируют решаемую систему линейных неравенств, могут либо быть конечными, но состоящими из очень большого числа элементов, либо представлять собой бесконечные последовательности, элементы которых появляются на входе распознающей системы в дискретном времени. Поэтому предпочтительны

итерационные методы (например, фейеровского типа), адаптирующиеся к условиям пополнения материала обучения в процессе его реализации [8, с. 138].

2.2.1 Методы линейной коррекции

Методы линейной коррекции имеют простейшую структуру. Рассмотрим задачу аффинного дискриминантного анализа: найти хотя бы одно решение $[x^*, y^*]$ системы:

$$\begin{aligned} &(x^*, x) + y^* > 0 \ Vx \in X_1 \\ &(x^*, x) + y^* < 0 \ Vx \in X_2 \end{aligned} \qquad (12)$$

Введем обозначения:

$$[x^*, y^*] = z^* \in R^n \ [x, 1] = z \in R^{n+1}$$

$Z\$ = \{ z = [x, 1] : x \in X\$ \}$, $(i = 1, 2)$ Тогда система примет вид:

$$\begin{aligned} &(z^*, z) > 0 \ (Vz \in Z_1) \\ &(z^*, z) < 0 \ (Vz \in Z'') \end{aligned} \qquad (13)$$

Из элементов множества Z_1 и Z_2 составляется последовательность $\{z_t\}$, в которой каждый элемент встречается бесконечное число раз, например, циклическая последовательность. Затем строится $\{z^*\}$ по формуле:

$$z^*_{t+1} = \quad + a + Z +$$

где a_t - коэффициент коррекции. В зависимости от правила выбора $a+$ получается тот или иной вид метода коррекции:

1) Алгоритм с постоянным коэффициентом коррекции: в нем $a_t = 0$, если

$$(z+*, z+) > 0, z+ \text{ £}$$

либо

$$(z+ z+) < 0, z+ \text{ £ } Z_2 \qquad (14)$$

В противном случае:

$$\begin{cases} d, \text{ если } z_t \text{ £ } Z_1 \\ -a, \text{ если } z_t \text{ £ } Z_2 \end{cases}$$

где $a > 0$;

2) Алгоритм абсолютной коррекции: полагаем $a_t = 0$, если выполняется соотношение (14); в противном случае:

$$a_t = \min [a: a > \frac{I(z*, z_t}{\quad}$$

3) Алгоритм частичной коррекции: полагаем $a_t = 0$, если выполняется (14), в противном случае a_t выбирается так, чтоб выполнялось равенство:

$$|(z*, z+) - (z*_{+1}, z+)| = A|(z*, z+)|$$

где A £ (0; 2).

2.2.2 Методы фейеровских отображений

Системе (12) можно поставить в соответствие систему нестрогих линейных неравенств

$$\begin{aligned} &(z^*, z) > 1 \ (Vz \in Z_{\pm}) \\ &(z^*, z) < 1 \ (Vz \in Z_2) \end{aligned} \qquad (15)$$

Если z* - решение системы (13), то Xz^* - решение системы (15) при достаточно большом A > 0; с другой стороны, всякое решение системы (15) удовлетворяет системе (13). Более того, решениями системы (13) могут быть приближенные решения системы (15): если

$$\begin{aligned} &(z^*, z) > 1 - \pounds \ (Vz \in Z_{\pm}) \\ &(z^*, z) < -1 + \pounds \ (Vz \in Z_2) \end{aligned}$$

То z^* - решение системы (13) при $\pounds < 1$. Следовательно, итерационный метод, дающий последовательность, которая сходится к решению системы (15), обеспечивает решение системы (13) за конечное число шагов. В качестве итерационного оператора может выступать фейеровское отображение [8, с. 140].

3 АЛГОРИТМ РЕШЕНИЯ ЗЛП С НЕФОРМАЛИЗОВАННЫМ ОГРАНИЧЕНИЕМ

На практике часто возникают ЗЛП с ограничениями, которые невозможно формализовать по тем или иным причинам. Очевидно, что в этом случае для решения задачи невозможно применить стандартные методы линейного программирования, описанные в главе 1, поэтому необходимо использовать другие методики и алгоритмы. Один из таких алгоритмов основан на использовании методов дискриминантного анализа (рис. 1).

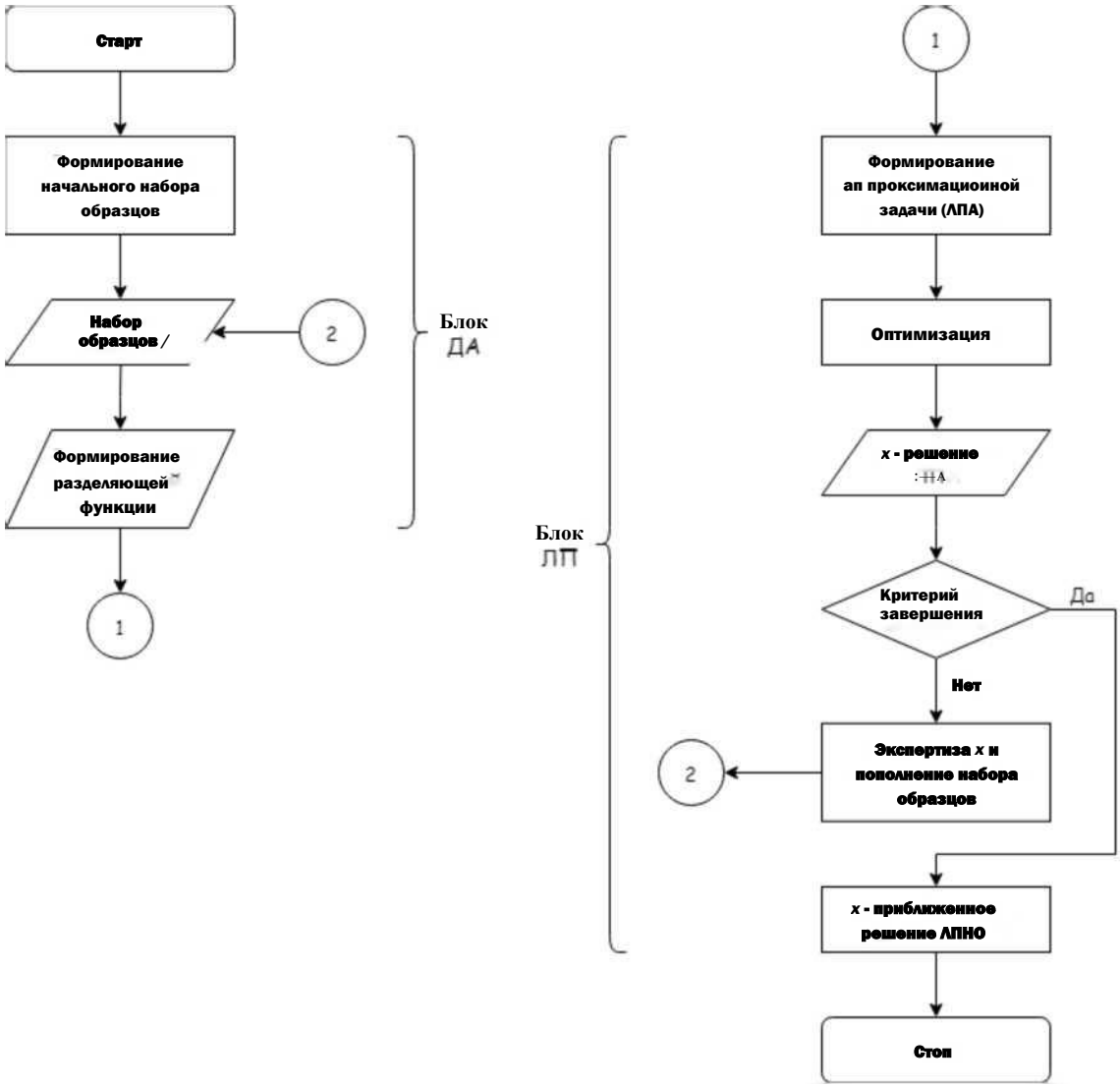

Рисунок 1 - Алгоритм решения ЗЛП с неформализованным ограничением

В блоке дискриминантного анализа происходит формирование начального набора образцов путем задания пары конечных непересекающихся множеств A и B из пространства R^n, удовлетворяющих условиям:

$$/(x) > 0 \ (\forall x \in A)$$
$$/(x) < 0 \ (\forall x \in B)$$

где /(x) - функция-эксперт.

На следующем шаге с помощью методов дискриминантного анализа строится аффинная функция $\phi(x)$, разделяющая множества A и B в соответствии с условиями (16).

Затем в блоке линейного программирования с использованием полученной разделяющей функции строится аппроксимационная задача линейного программирования:

$$L(X) = \wedge c\%x\% \wedge \max(min)_{j=1}^{n}$$
$$\wedge a\$\%X_j < b\$, i = \overline{l,r}$$
$$\%=1$$
$$\wedge(x) < 0$$
$$X_j > 0, j = \overline{1, n}$$

для решения которой может использоваться любой метод линейного программирования.

Полученное решение x, с помощью некоторого критерия завершения итерационного процесса, проверяется на близость к точному решению x.

В случае, если решение не удовлетворяет критерию завершения, оно отправляется в блок экспертизы, где классифицируется функцией-экспертом $f(x)$

- если $f(x) > 0$, то решение добавляется в множество A, а если $f(x) < 0$ - в множество B. После этого описанный процесс повторяется применительно к расширенному набору образцов.

Следует отметить, что при проведении экспертизы и пополнении набора образцов могут возникать следующие коллизии:

1) $f\{x) = 0$;
2) $f(x) > 0$ и x е A;
3) $fix) < 0$ и x e B.

В каждом из этих случаев к x применяется процедура рандомизации [25]. Точки, полученные в результате процедуры рандомизации, вновь подвергаются экспертизе, и используются для пополнения образцов.

4 РЕАЛИЗАЦИЯ АЛГОРИТМА РЕШЕНИЯ ЗЛП С НЕФОРМАЛИЗОВАННЫМ ОГРАНИЧЕНИЕМ НА БАЗЕ КЛИЕНТСЕРВЕРНОГО ПРИЛОЖЕНИЯ

Реализация алгоритма решения ЗЛП с неформализованным ограничением, представленного в 3 главе, требует решения следующих задач:

1) Формирование начального набора образцов;
2) Выбор критерия завершения;
3) Выбор метода формирования разделяющей функции;
4) Выбор метода решения аппроксимационной ЗЛП;
5) Выбор среды разработки.

4.1 Формирование начального набора образцов

Начальный набор образцов задается пользователем при запуске алгоритма.

4.2 Выбор критерия завершения работы алгоритма

В качестве критерия завершения работы алгоритма используется условие вида [25]:

$$\sum_{i=1}^{2} \| * - \%_k - i \| < £$$

где e > 0 - заданное вещественное число, определяющее точность вычислений.

Формально это значит, что алгоритм прекратит свою работу в том случае, если сумма расстояний между $к$-$ым$ решением задачи и $к$ — 1, и $к$ — 1 и $к$ — 2 будет меньше £.

4.3 Выбор метода формирования разделяющей функции

В качестве метода формирования разделяющей функции используется метод линейной коррекции в силу простоты реализации данного метода.

4.4 Выбор метода решения аппроксимационной ЗЛП

В качестве метода решения аппроксимационной ЗЛП используется симплексметод в силу универсальности данного метода.

4.5 Выбор среды разработки

В качестве среды разработки алгоритма используется Vue.js (реализация клиентской части) и Node.js (реализация серверной части) в силу имеющегося опыта разработки с использованием данных технологий.

4.6 Реализация алгоритма на базе приложения

После запуска приложения пользователь увидит на экране 2 «зоны»:

1) Зона ввода исходных данных;
2) Зона найденного решения.

Зона ввода исходных данных представляет собой набор полей для загрузки модели ЗЛП в виде файла[1] с расширением .aslppucid (поле «Model»), ввода точности (поле «Accuracy») и параметров рандомизации - радиуса и точности (поля «Randomization radius» и «Randomization power» соответственно) (рис. 2).

[1] **Исходный код приложения и пример файла, содержащего модель ЗЛП можно найти в репозитории https://github.com/inikonorov/aslppucid-2.**

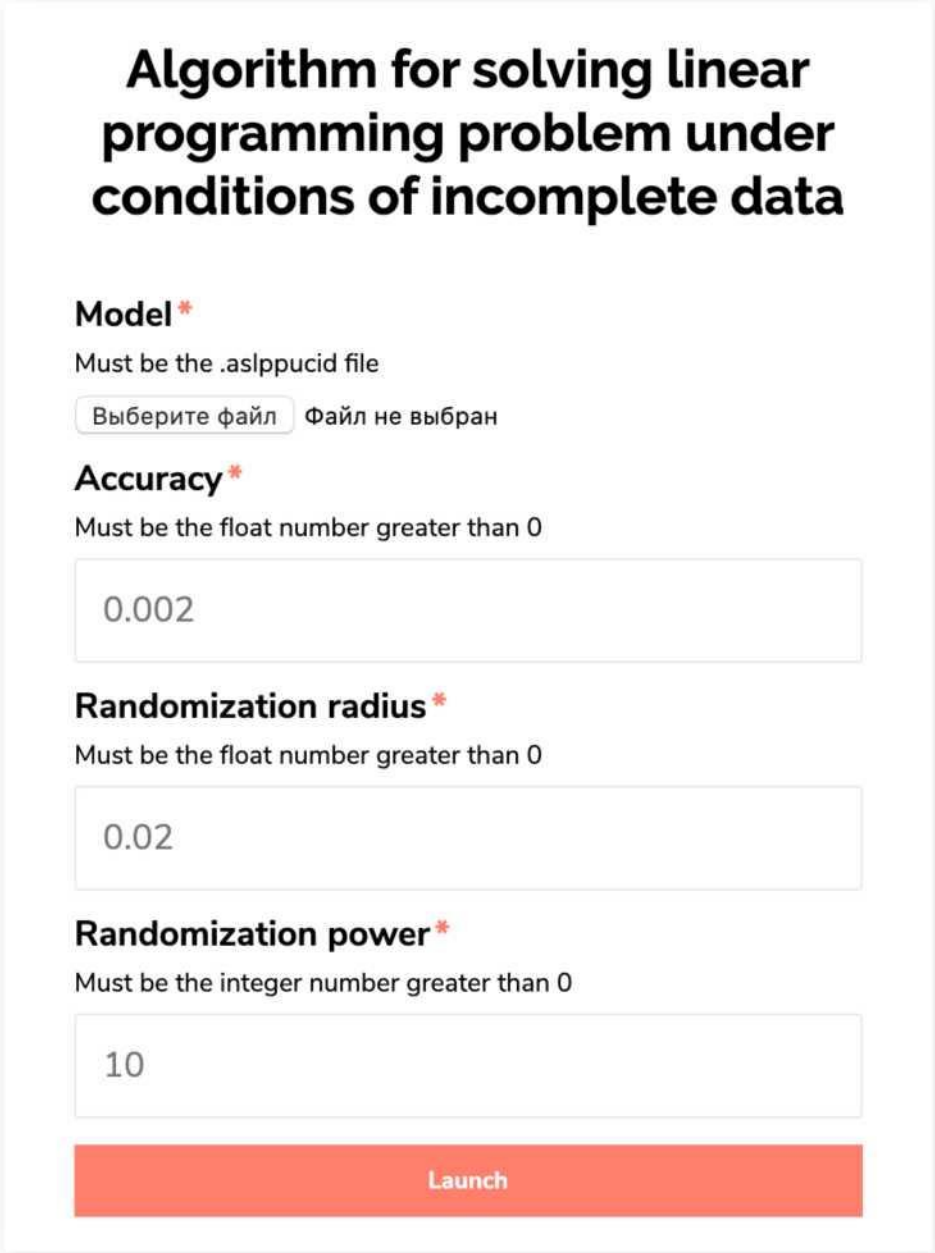

Рисунок 2 - Зона ввода исходных данных

Зона найденного решения представляет собой текстовый блок и может находиться в двух состояниях (рис 3-4).

Solution will be here!

Рисунок 3 - Состояние зоны найденного решения перед началом работы алгоритма

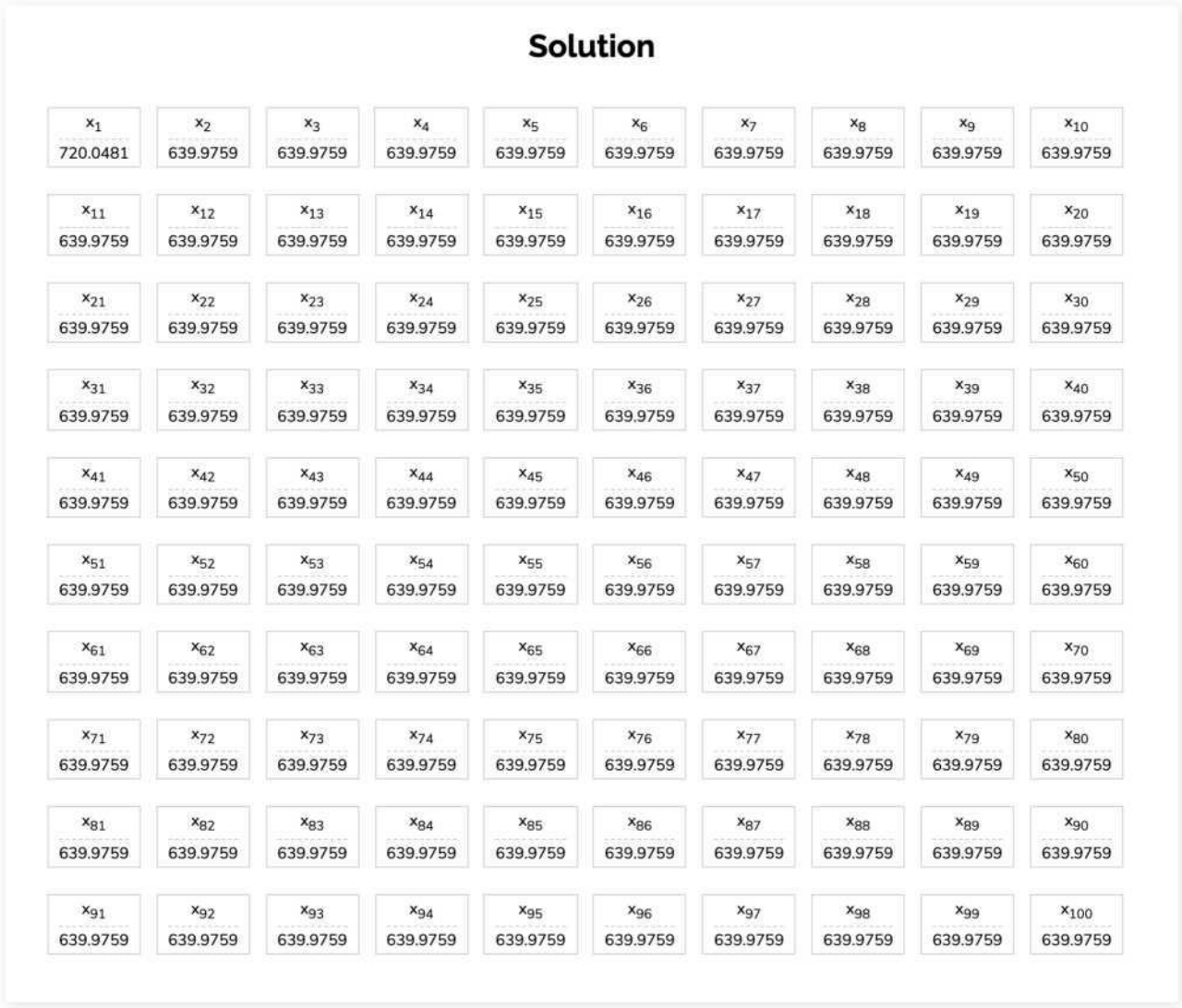

Рисунок 4 - Состояние зоны найденного решения после окончания работы алгоритма

После того, как пользователь заполнил все поля и нажал кнопку «Launch», запрос с данными, которые ввел пользователь, отправляются на сервер, где с ними начинает работать непосредственно сам алгоритм - он извлекает из файла с моделью необходимые данные о ЗЛП (целевую функцию, ограничения, точки множеств), строит разделяющую функцию, после чего составляет с ней аппроксимационную ЗЛП, решает ее и проверяет полученное решение на предмет выполнения критерия завершения - если критерий не выполняется, то алгоритм передает полученное решение на экспертизу, после чего запускается снова и снова до тех пор пока не выполнится критерий завершения.

5 ИССЛЕДОВАНИЕ РАБОТЫ АЛГОРИТМА

Исследование работы алгоритма проводится на модельной задаче размерности $n <$ 100:

$$Q = x_{\pm} \wedge max$$

$$\text{г} \quad x_{\pm} — 2x_2 < 0$$

$$x_{\pm} — 2x_3 < 0$$

$$x_{\pm} — 2x_n < 0 <$$

$$x_{\pm} + 2x_2 < 2000$$

$$x_{\pm} + 2x_3 < 2000$$

$$x_{\pm} + 2x_n < 2000$$

$$< x_{\pm} > 0, x_2 > 0, \ldots x_n > 0$$

$$\wedge(x) = 2\ (n — 1)x_{\pm} — x_2 — x_3 — \text{------} x_n — 800(n — 1)$$

Функция $\phi(x)$ моделирует эксперта в том смысле, что она явно задает неформализованное ограничение $\phi(x) < 0$. Для всех размерностей $n > 2$ эта задача имеет следующее единственное точное решение x: $x\wedge = 720$, $x\wedge = \bullet\bullet\bullet = x\# = 640$. В качестве начального набора образцов используется множество всех вершин выпуклого многогранника, задаваемого формализованными ограничениями задачи:

$$A = \{(1000; 500; \ldots,500)\}$$

$$B = \{(0; \ldots; 0), (0; 1000; 0; \ldots; 0),\ldots, (0; \ldots; 0; 1000), (0; 1000; .;1000)\}$$

При этом |Л| = 1, |5| = n + 1.

5.1 Анализ выбора £ в зависимости от размерности задачи

В теории, можно запустить алгоритм решения задачи любой размерности для любой точности, но следует учитывать тот факт, что чем точнее решение необходимо найти, тем дольше будет работать алгоритм. В какой-то момент стоимость операций по получению очередного знака после запятой превысит стоимость времени, которое требуется для решения задачи. Чтобы избежать такой ситуации необходимо задать некоторые рамки допустимой точности, на основании которой будет формироваться £ критерия завершения работы алгоритма. Считается, что приемлемым отклонением от истинного решения являются значения из диапазона 0.001-0.1.

Тогда, в соответствии с выбранным критерием завершения

$$(\text{ll-}\wedge_{\mathbf{k}} \quad \text{-}\wedge_{\mathbf{k\text{-}l}}\text{Hmin}) + (\text{ll-}\wedge_{\mathbf{k\text{-}l}} \%\kappa\text{-}2\ \text{Hmin}) < \wedge$$

$$< (\text{l}\backslash\wedge_{\mathbf{-k}} \quad \wedge_{\mathbf{-k\text{-}l}} \text{II}\ max\ ) + (\backslash\backslash Xk\text{-}i\ \text{-}x,\ \text{-}2\ \backslash\mathbf{max})$$

$$\wedge : V(\wedge kj \quad \%(k\text{-}l)i)\text{min}) + \underset{i=1}{\overset{n}{\wedge}}((^{g}(k\sim l)i \quad k\text{-}2)i)\text{min}) <$$

$$N \qquad\qquad\qquad N$$

$$< \frac{\wedge\wedge((\wedge\text{-ki} \quad X(k\text{-}l)i)\text{max})}{Л1^{i=1}}^{n} + \frac{\wedge\ {}^{<}(O^{g}(k\text{-}l)j \quad \text{-}^{g}(k\text{-}2)i)\text{max})}{Л1^{i=1}}^{n}$$

где

$$— \text{-}^{g}(k\text{-}l)i)\text{min} - (\text{-}^{g}(k\text{-}l)i — \text{-}^{g}(k\text{-}2)i)\text{min} - °.^{001},$$

$$— \text{-}^{g}(k\text{-}l)i)\text{max} - (\text{-}^{g}(k\text{-}l)i — \text{-}^{g}(k\text{-}2)i)\text{max} - °.\wedge$$

Следовательно

$$0.00\text{lVn} + 0.00\text{lVn} < £ < O.\text{lVn} + O.\text{lVn}$$

$$0.002VH < £ < 0.2Vft$$

Вычисления в главах 5.2, 5.3 производятся на задачах размерности $n = 20; 40; 60; 80; 100$, допустимые значения точности £ которых указаны в таблице 2.

Таблица 2 - Минимальное, максимальное и среднее допустимые значения точности £

	min £	max£	Среднее
20	0.0089	0.89	0.449
40	0.0126	1.26	0.636
60	0.0155	1.55	0.783
80	0.0179	1.79	0.901
100	0.02	2	1.01

5.2 Влияние радиуса рандомизации на эффективность алгоритма

Исследование влияния радиуса рандомизации на эффективность алгоритма проводится при следующих исходных данных:

1) Радиус рандомизации принимает значение $p = 0.001; 0.005; 0.01; 0.05; 0.1$;
2) Мощность рандомизации принимает значение $m = 5$;
3) Точность работы алгоритма принимает среднее значение в соответствии с таблицей 3.

На рис. 5-9 представлены зависимости времени работы алгоритма от радиуса рандомизации для задач размерностей $n = 20; 40; 60; 80; 100$.

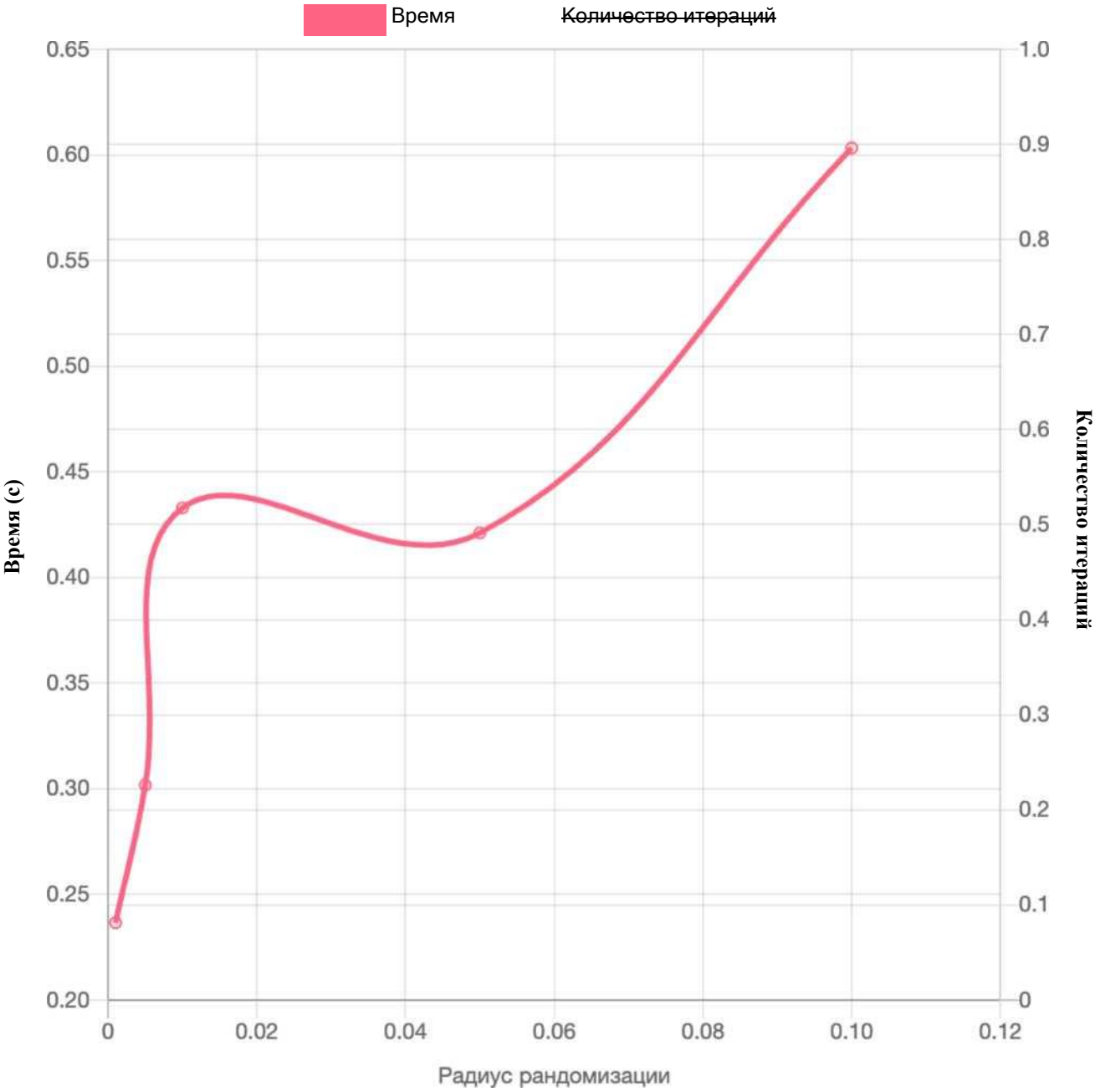

Рисунок 5 - Зависимость времени работы алгоритма от радиуса рандомизации при решении задачи размерности $n = 20$

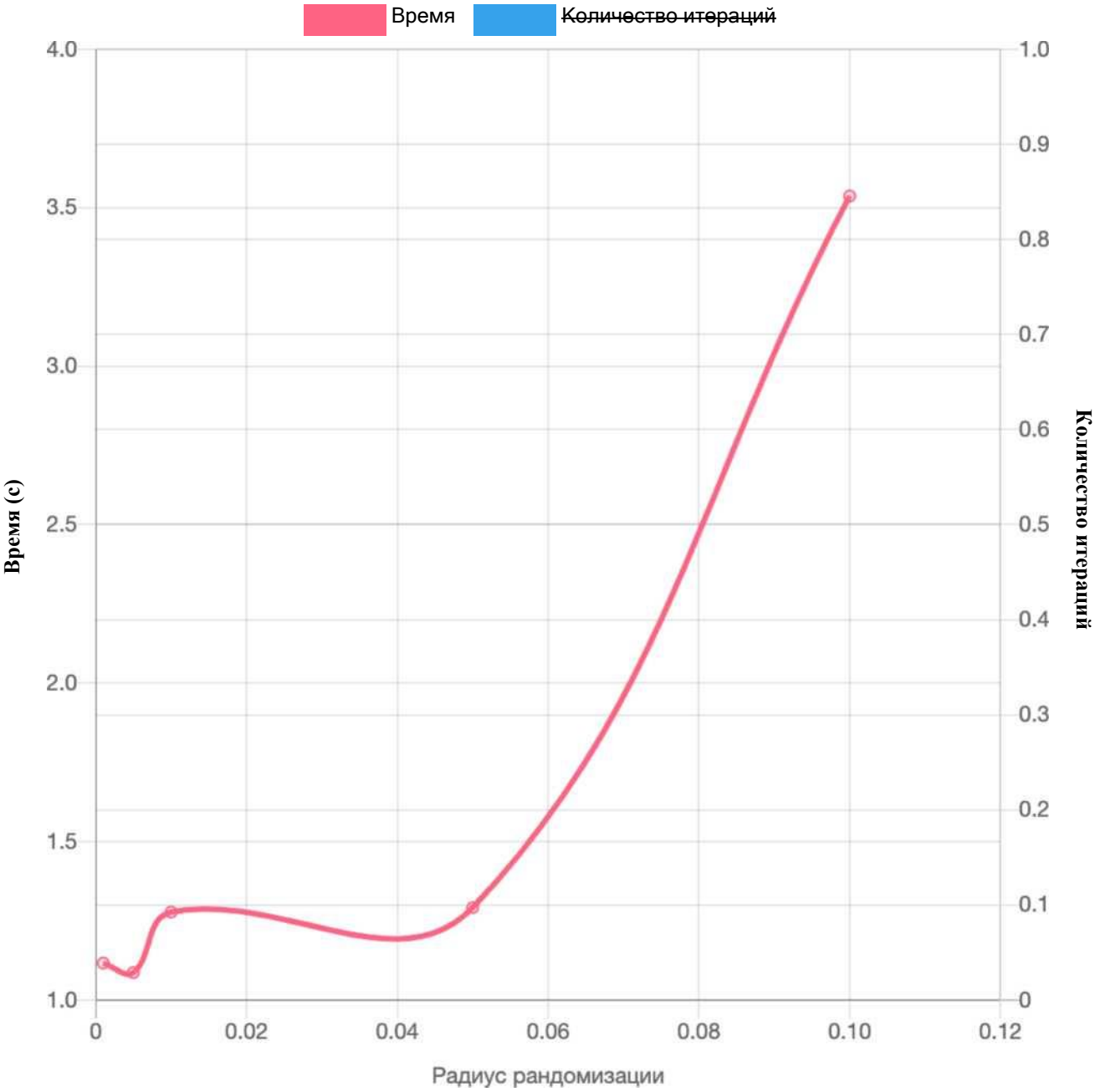

Рисунок 6 - Зависимость времени работы алгоритма от радиуса рандомизации при решении задачи размерности $n = 40$

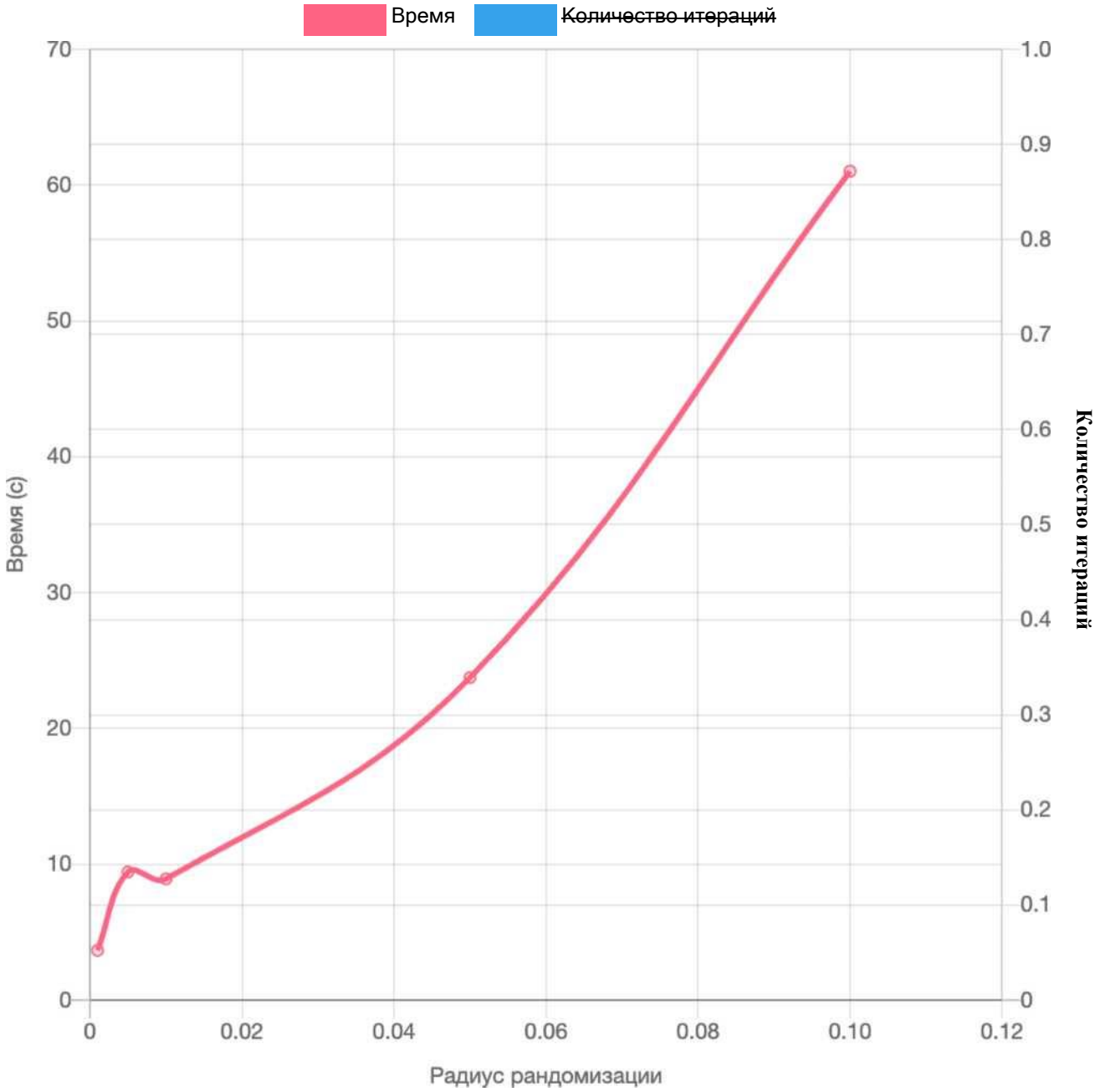

Рисунок7 - Зависимость времени работы алгоритма от радиуса рандомизации при решении задачи размерности $n = 60$

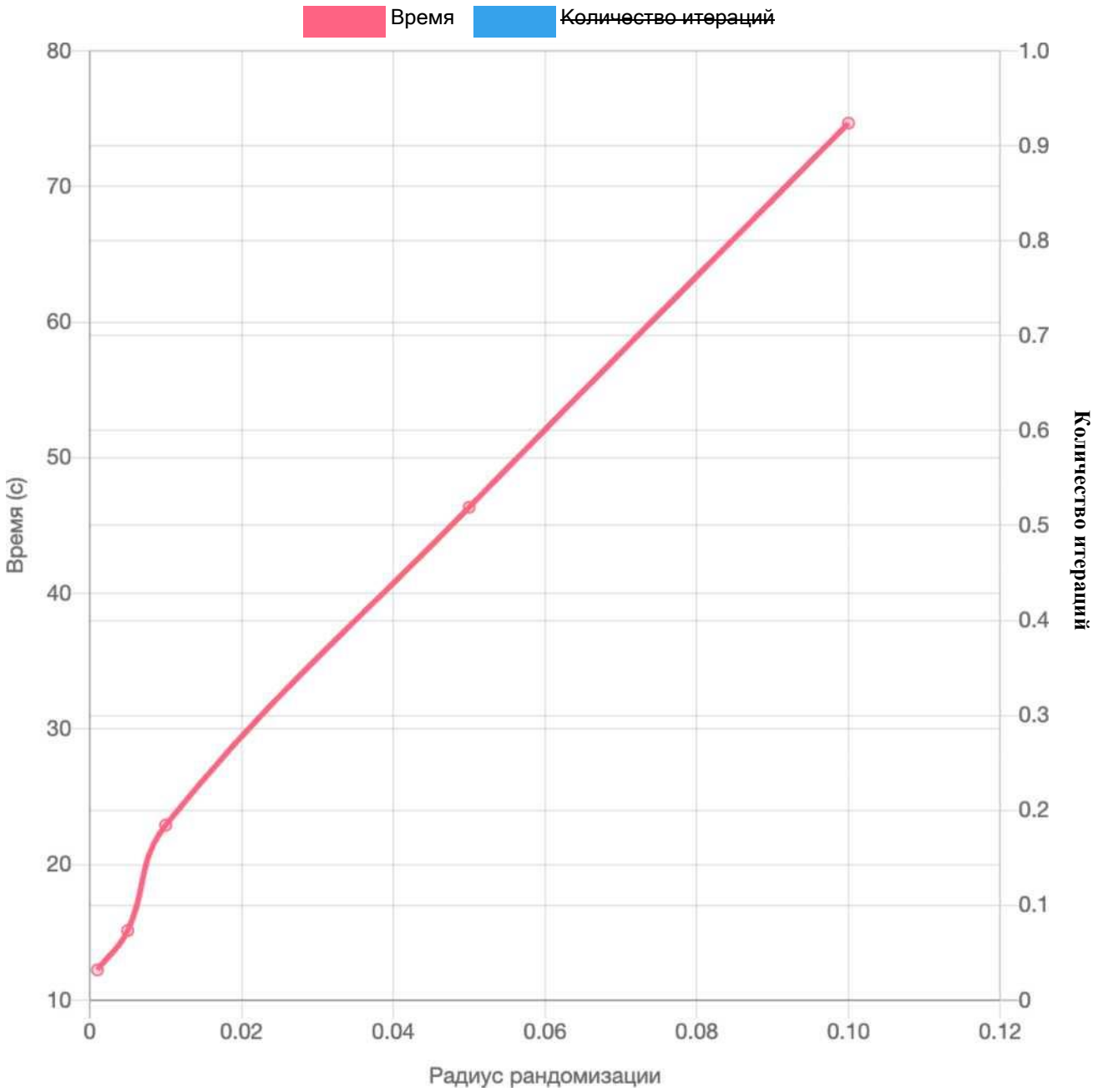

Рисунок 8 - Зависимость времени работы алгоритма от радиуса рандомизации при решении задачи размерности $n = 80$

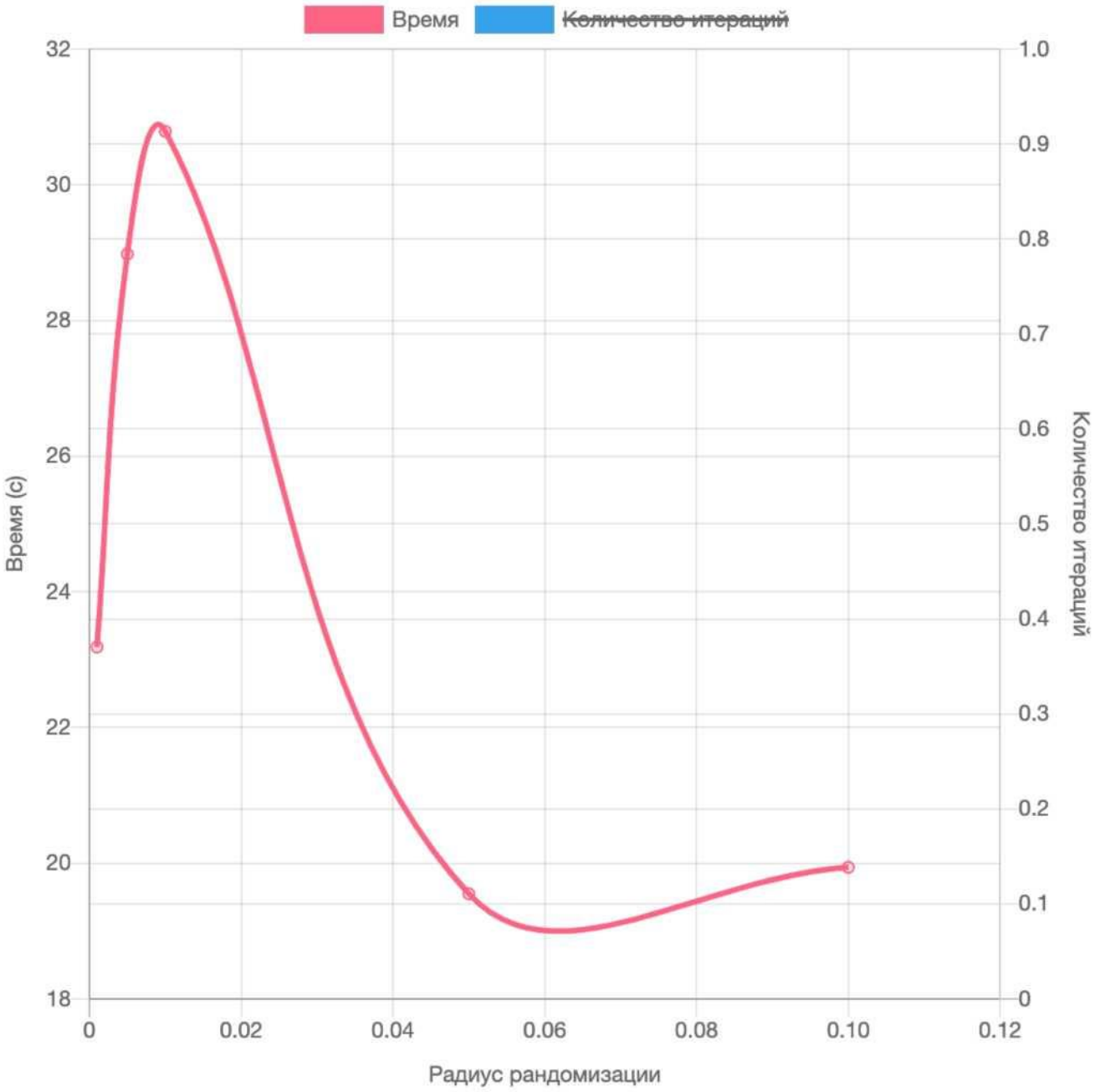

Рисунок 9 - Зависимость времени работы алгоритма от радиуса рандомизации при решении задачи размерности $n = 100$

Зависимости, представленные на рис. 5-8 носят однородный характер - при увеличении радиуса рандомизации время работы алгоритма увеличивается. При решении задачи размерности $n = 100$ (рис. 9) была выявлена закономерность другого рода; в этом случае зависимость содержит точки экстремума - $p \sim 0.01$

(наибольшее время работы алгоритма) и $p \sim 0.06$ (наименьшее время работы алгоритма).

На рис. 10-14 представлены зависимости количества итераций алгоритма от радиуса рандомизации для задач размерностей $n = 20; 40; 60; 80; 100$.

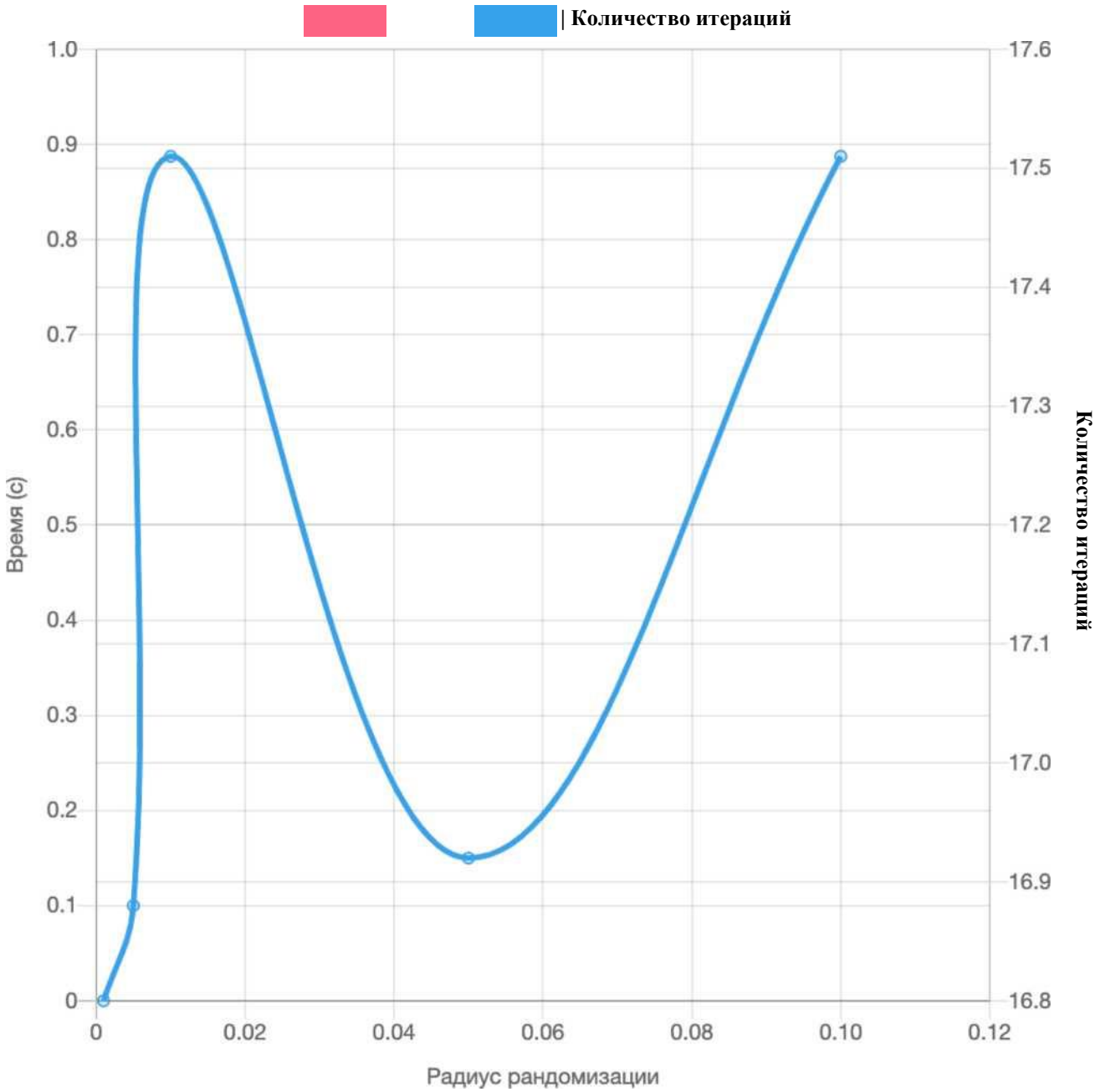

Рисунок 10 - Зависимость количества итераций алгоритма от радиуса рандомизации при решении задачи размерности $n = 20$

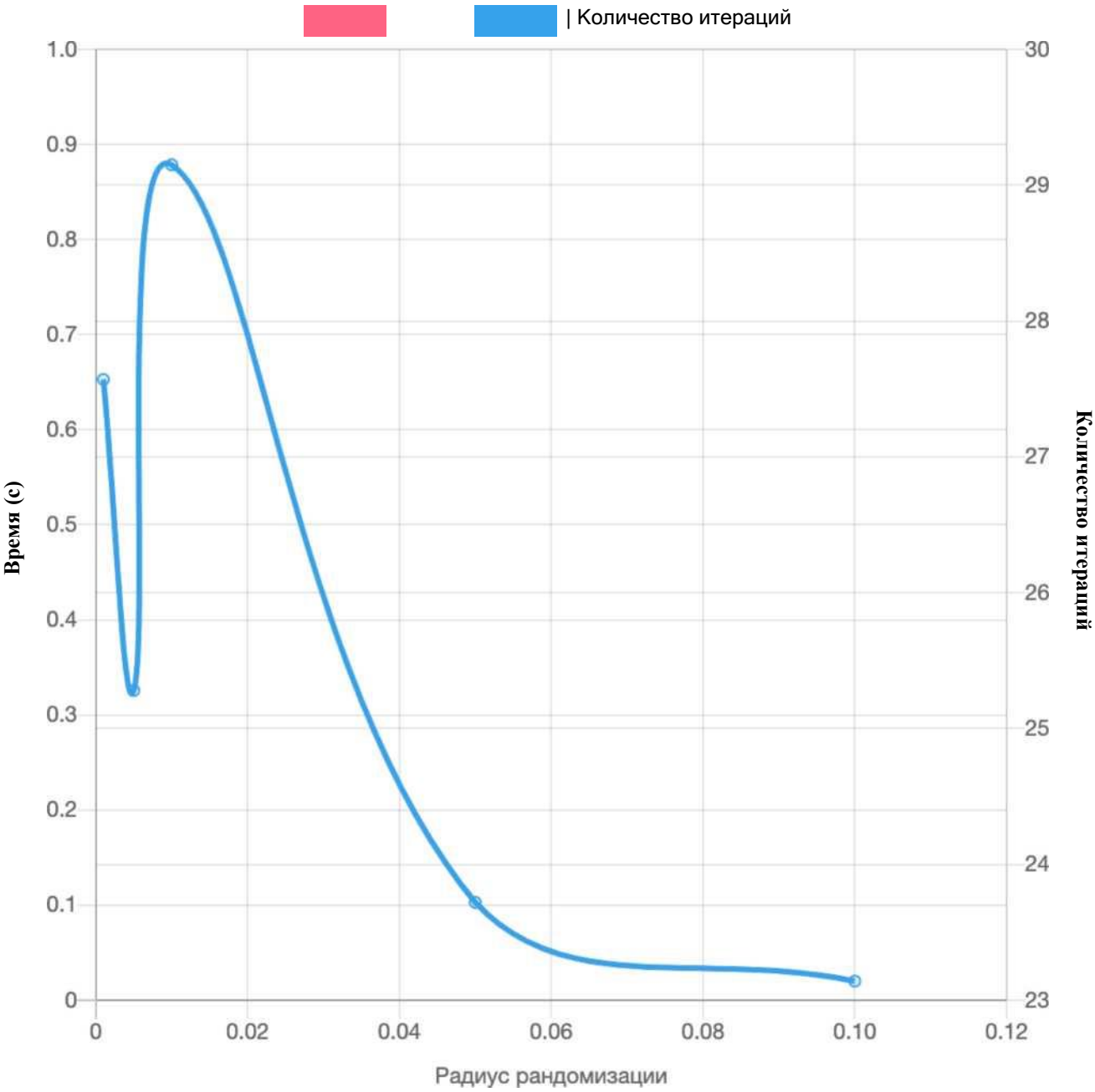

Рисунок 11 - Зависимость количества итераций алгоритма от радиуса рандомизации при решении задачи размерности $n = 40$

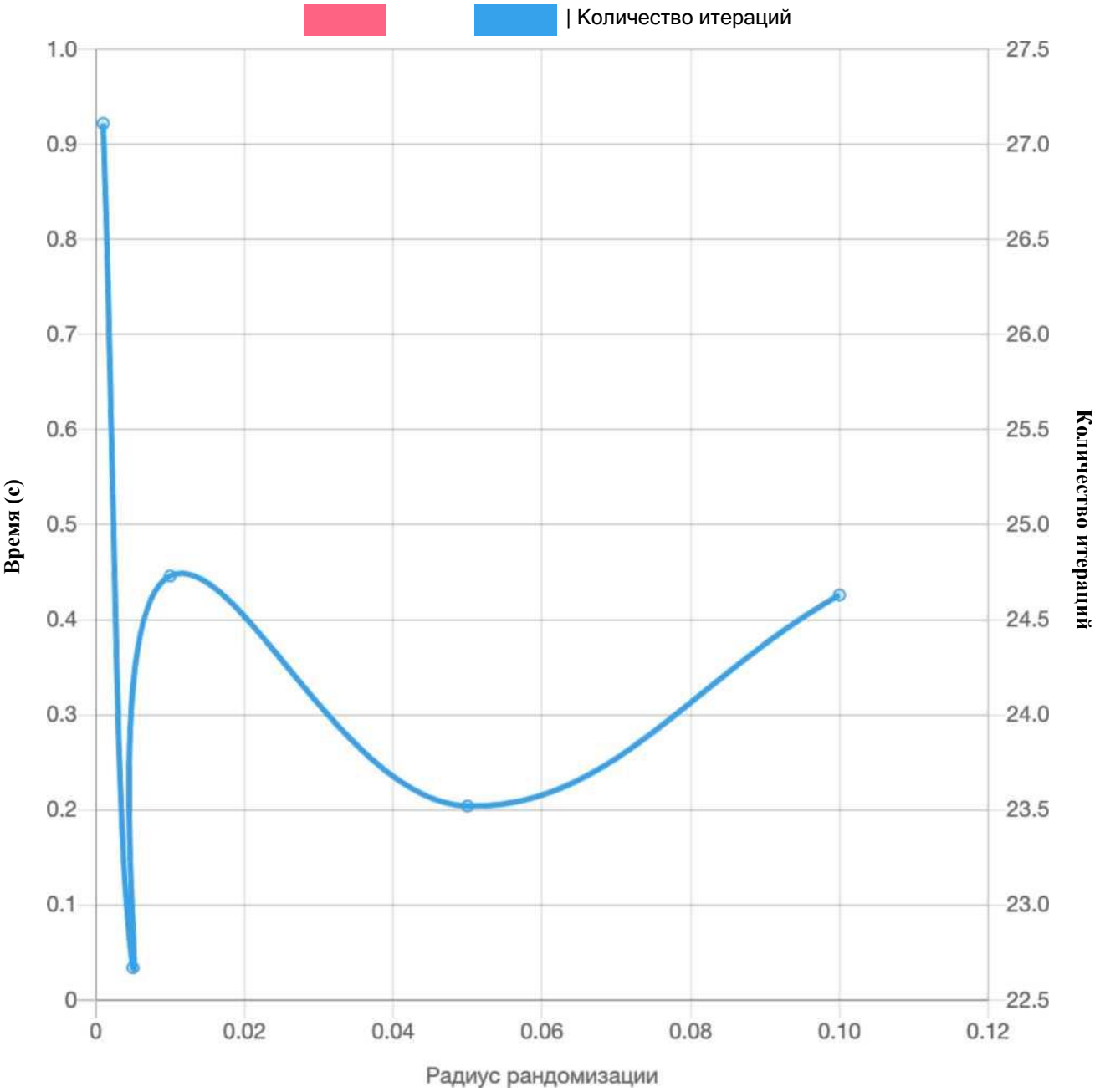

Рисунок 12 - Зависимость количества итераций алгоритма от радиуса рандомизации при решении задачи размерности $n = 60$

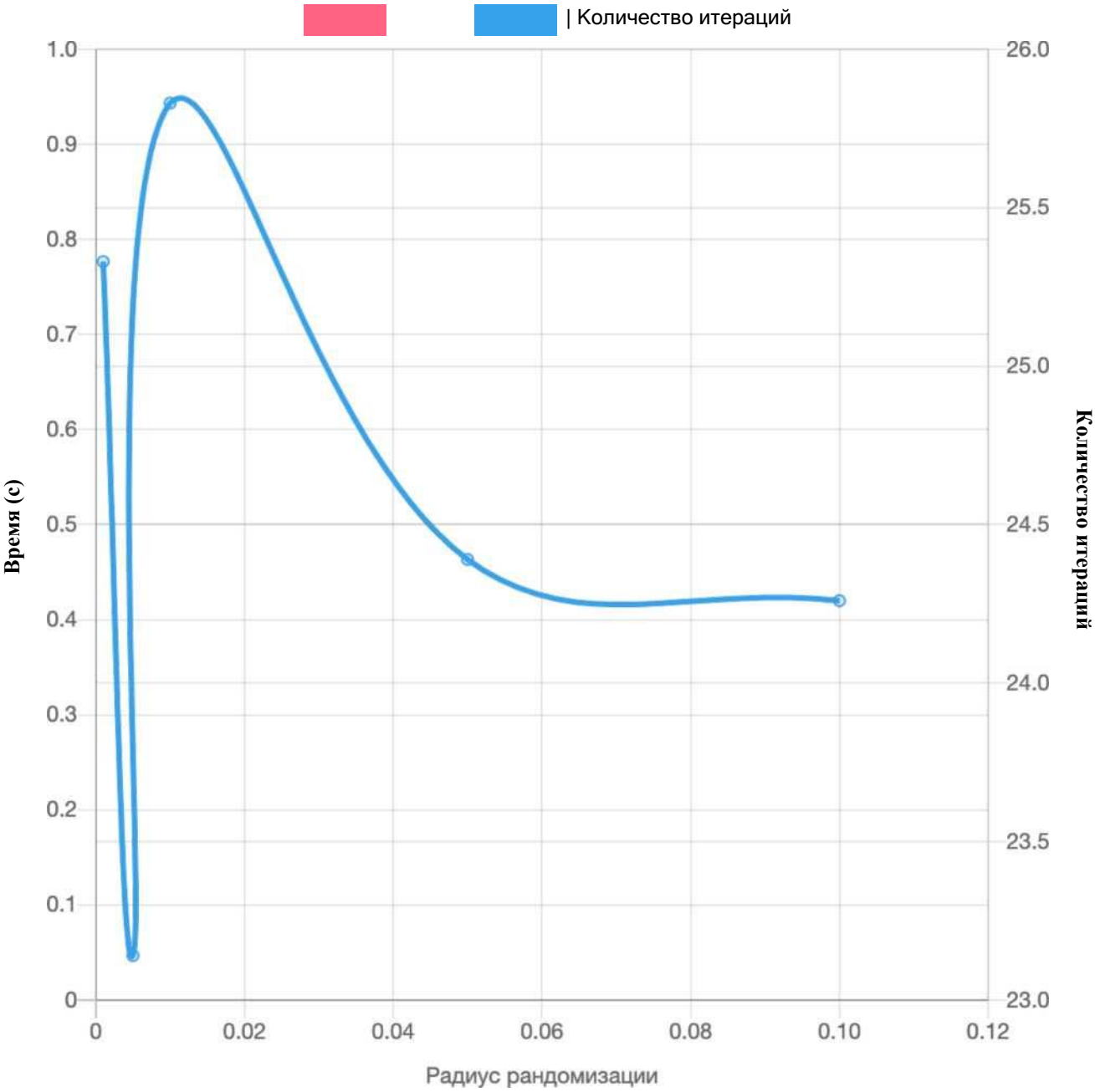

Рисунок 13 - Зависимость количества итераций алгоритма от радиуса рандомизации при решении задачи размерности $n = 80$

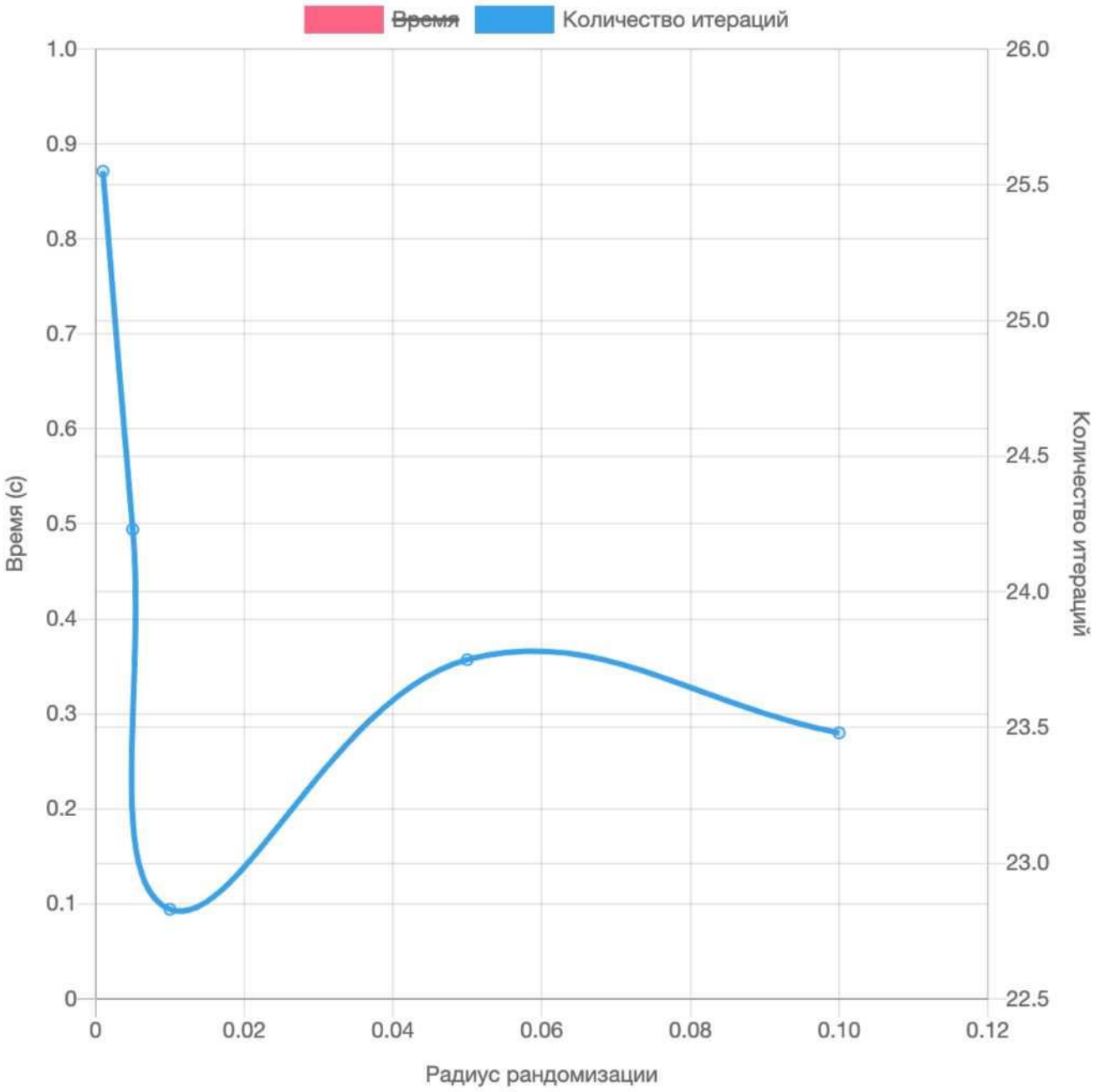

Рисунок 14 - Зависимость количества итераций алгоритма от радиуса рандомизации при решении задачи размерности $n = 100$

Зависимости, представленные на рис. 11-14 носят однородный характер - при увеличении радиуса рандомизации количество итераций алгоритма стремится к среднему значению (~ 24). При решении задачи размерности $n = 20$ (рис. 10) была выявлена закономерность другого рода; в этом случае количество итераций практически не зависит от радиуса рандомизации.

5.3 Влияние мощности рандомизации на эффективность алгоритма

Исследование влияния мощности рандомизации на эффективность алгоритма проводится при следующих исходных данных:

1) Мощность рандомизации принимает значение $m = 1; 5; 10; 50; 100;$
2) Радиус рандомизации принимает значение $p = 0.01;$
3) Точность работы алгоритма принимает среднее значение в соответствии с таблицей 3.

На рис. 14-18 представлены зависимости времени работы алгоритма от мощности рандомизации для задач размерностей $n = 20;$ 40; 60; 80; 100.

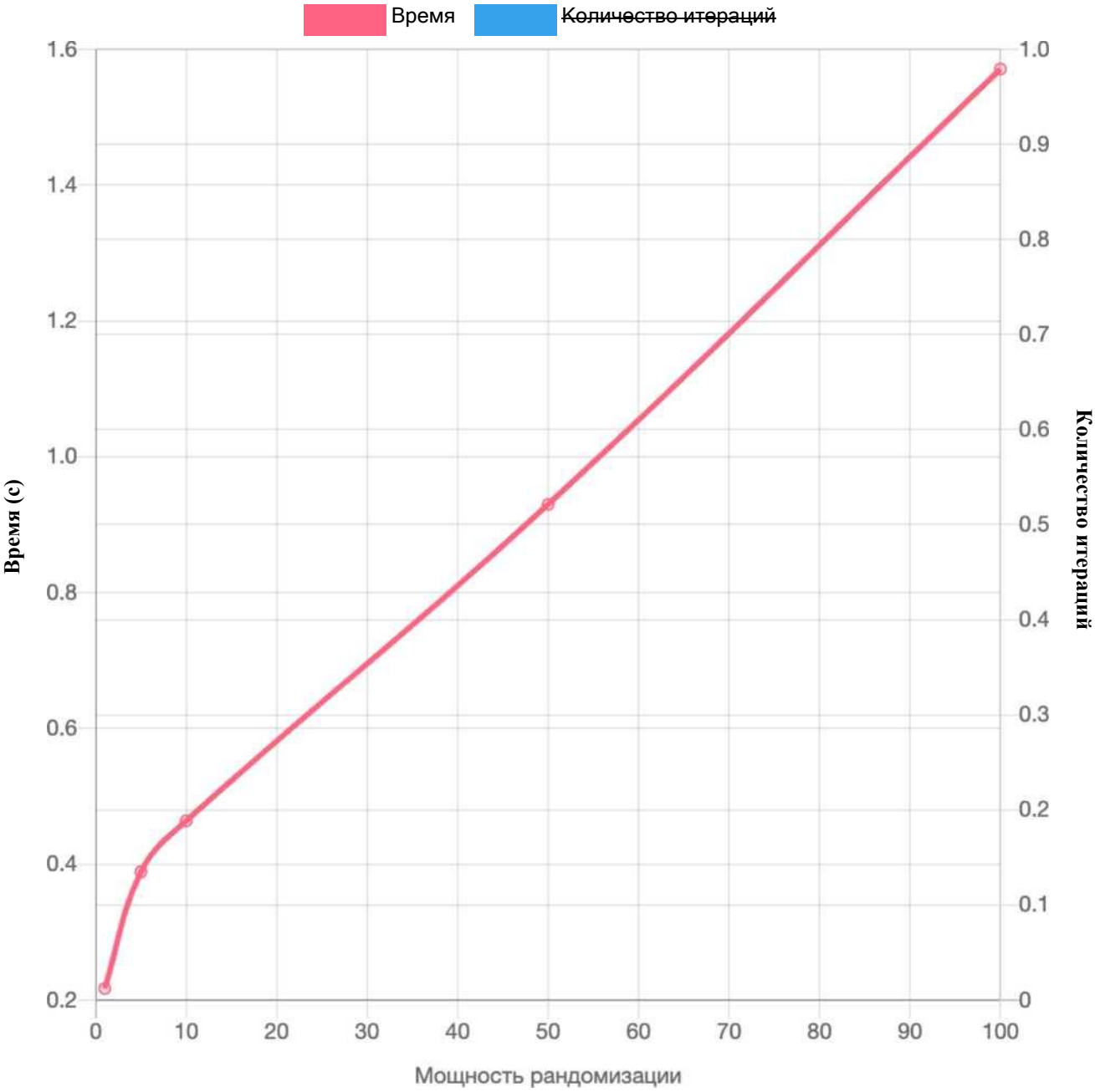

Рисунок 14 - Зависимость времени работы алгоритма от мощности рандомизации при решении задачи размерности $n = 20$

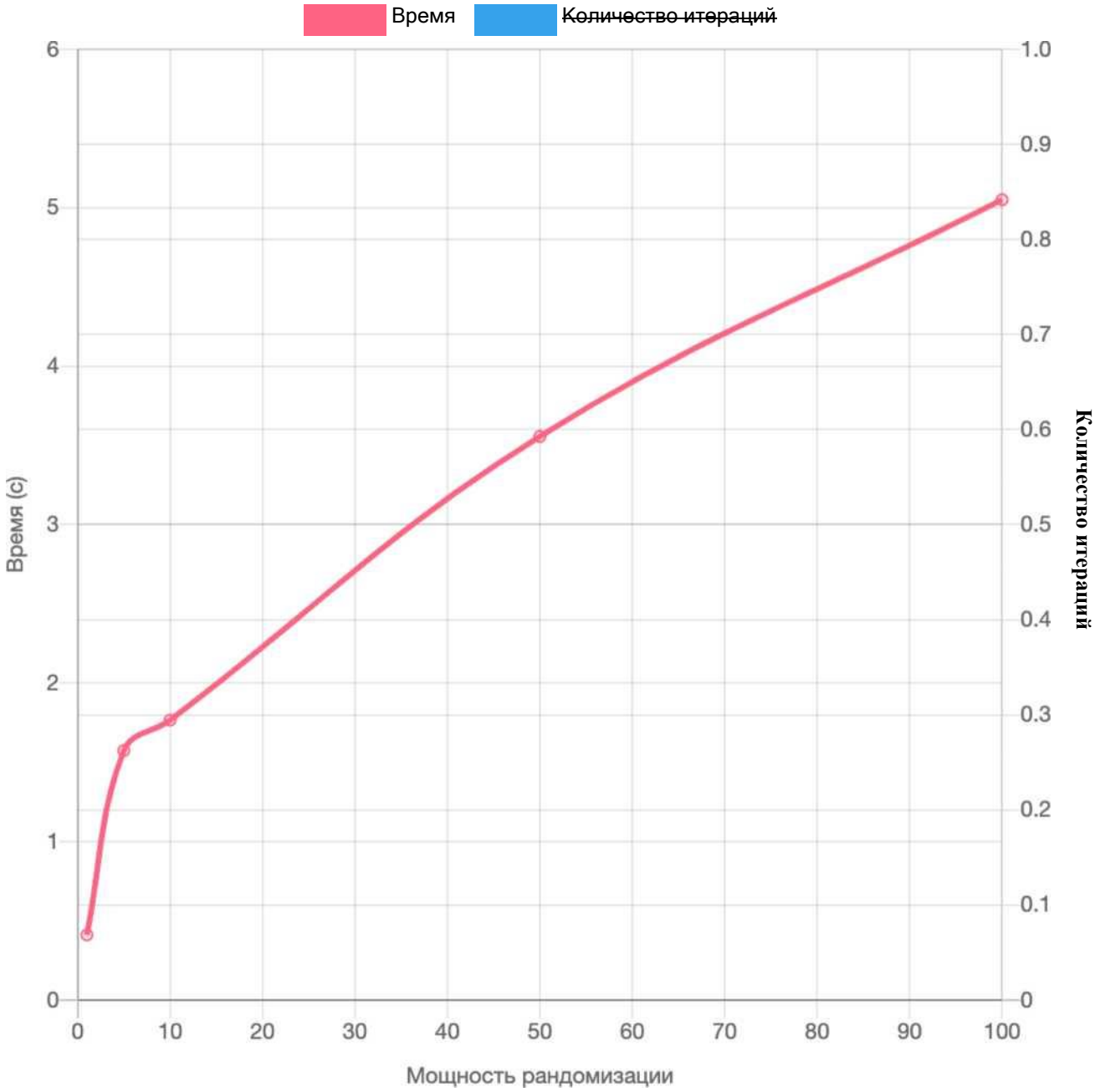

Рисунок 15 - Зависимость времени работы алгоритма от мощности рандомизации при решении задачи размерности $n = 40$

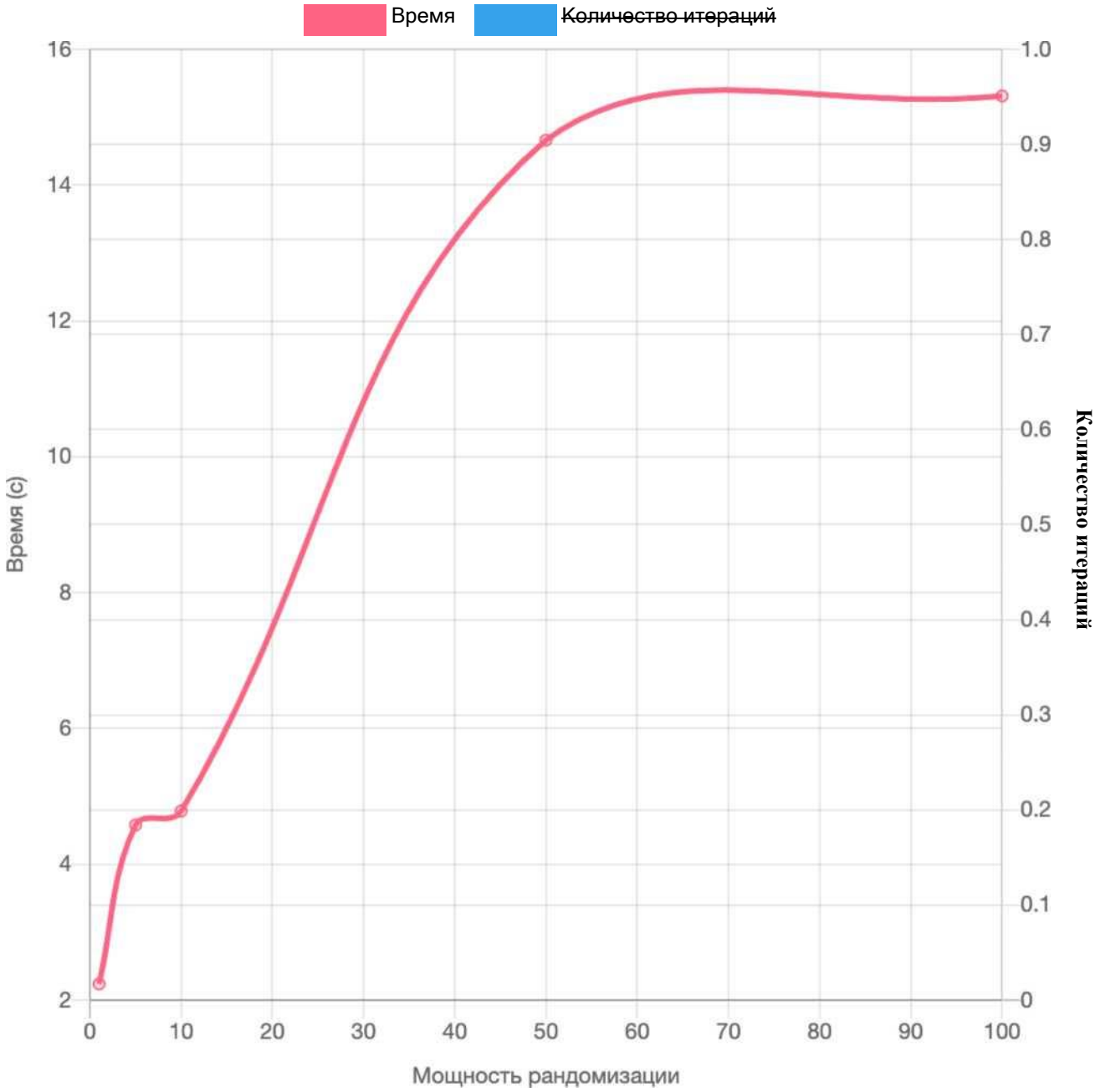

Рисунок 16 - Зависимость времени работы алгоритма от мощности рандомизации при решении задачи размерности $n = 60$

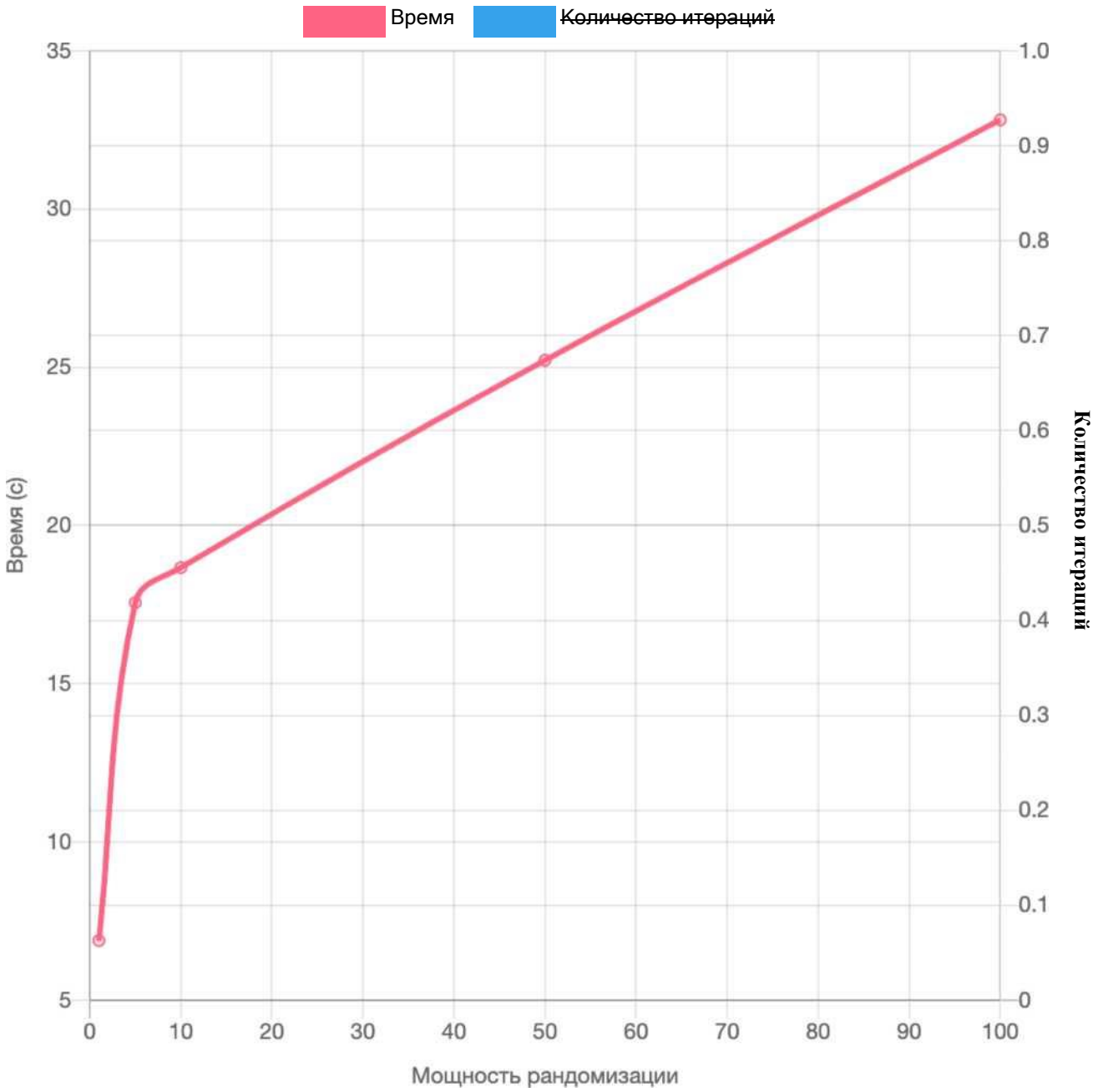

Рисунок 17 - Зависимость времени работы алгоритма от мощности рандомизации при решении задачи размерности $n = 80$

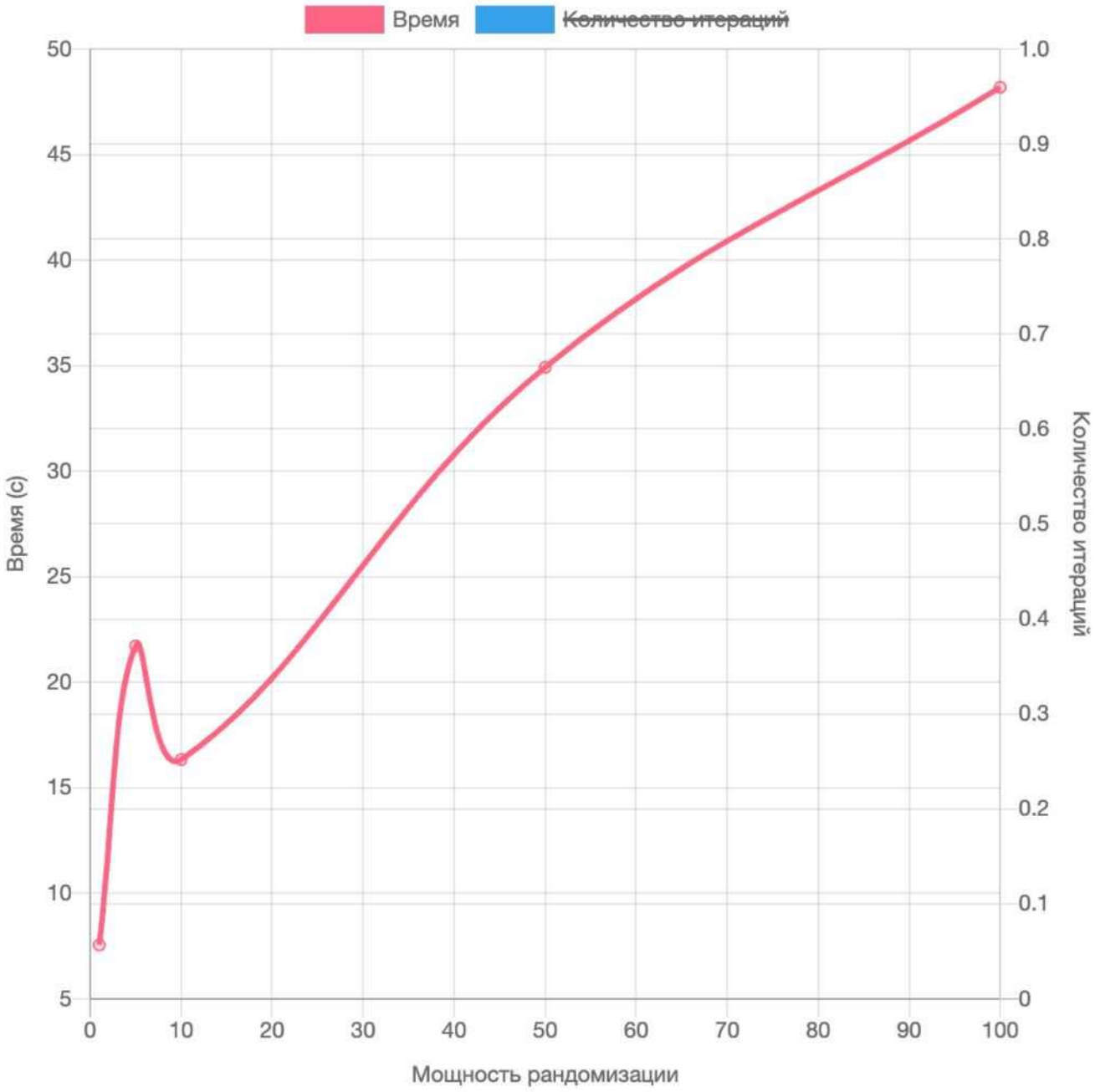

Рисунок 18 - Зависимость времени работы алгоритма от мощности рандомизации при решении задачи размерности $n = 100$

Зависимости, представленные на рис. 14-18 носят однородный характер - при увеличении мощности рандомизации время работы алгоритма увеличивается.

На рис. 19-23 представлены зависимости количества итераций алгоритма от мощности рандомизации для задач размерностей $n = 20; 40; 60; 80; 100$.

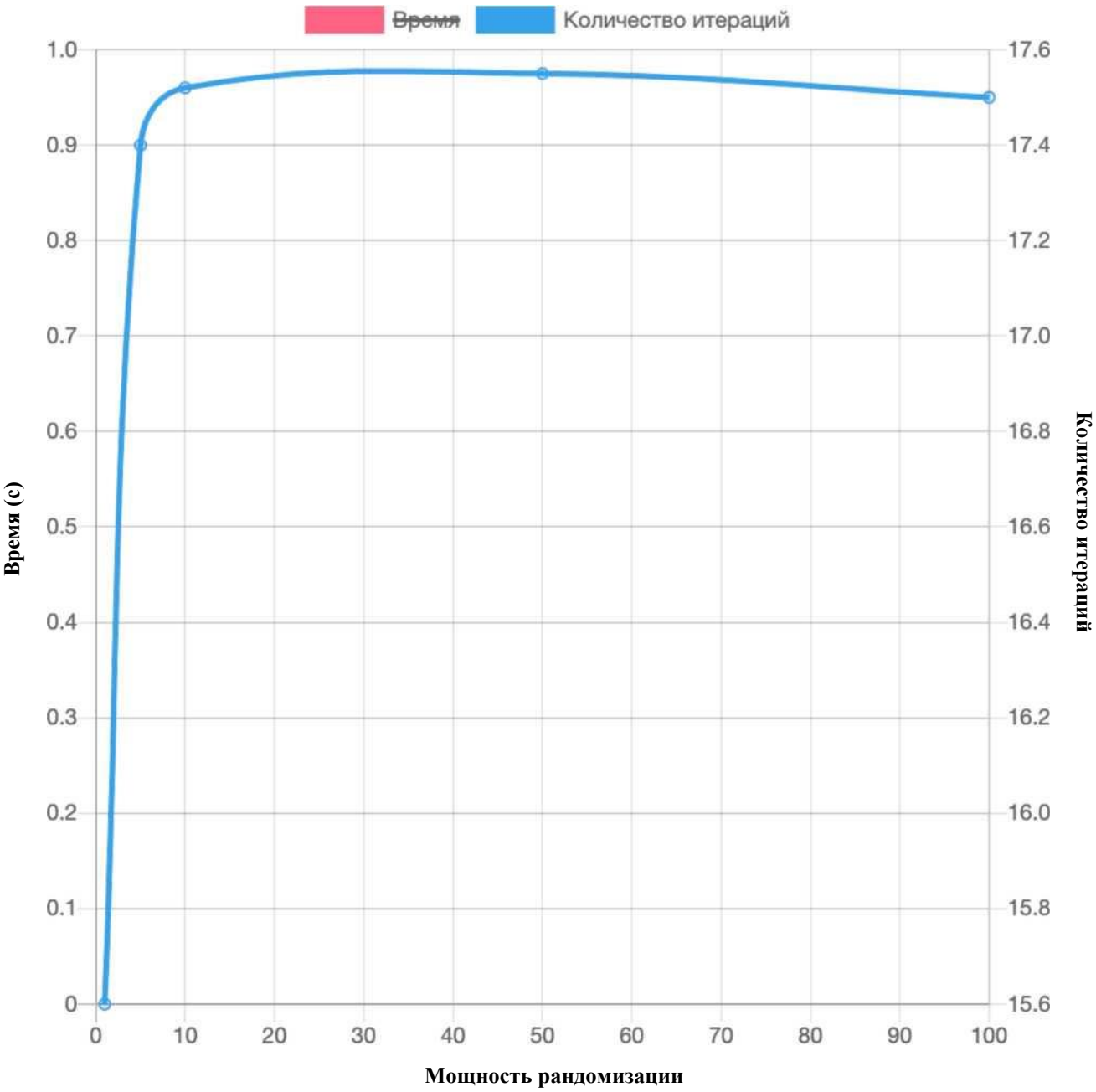

Рисунок 19 - Зависимость количества итераций алгоритма от мощности рандомизации при решении задачи размерности $n = 20$

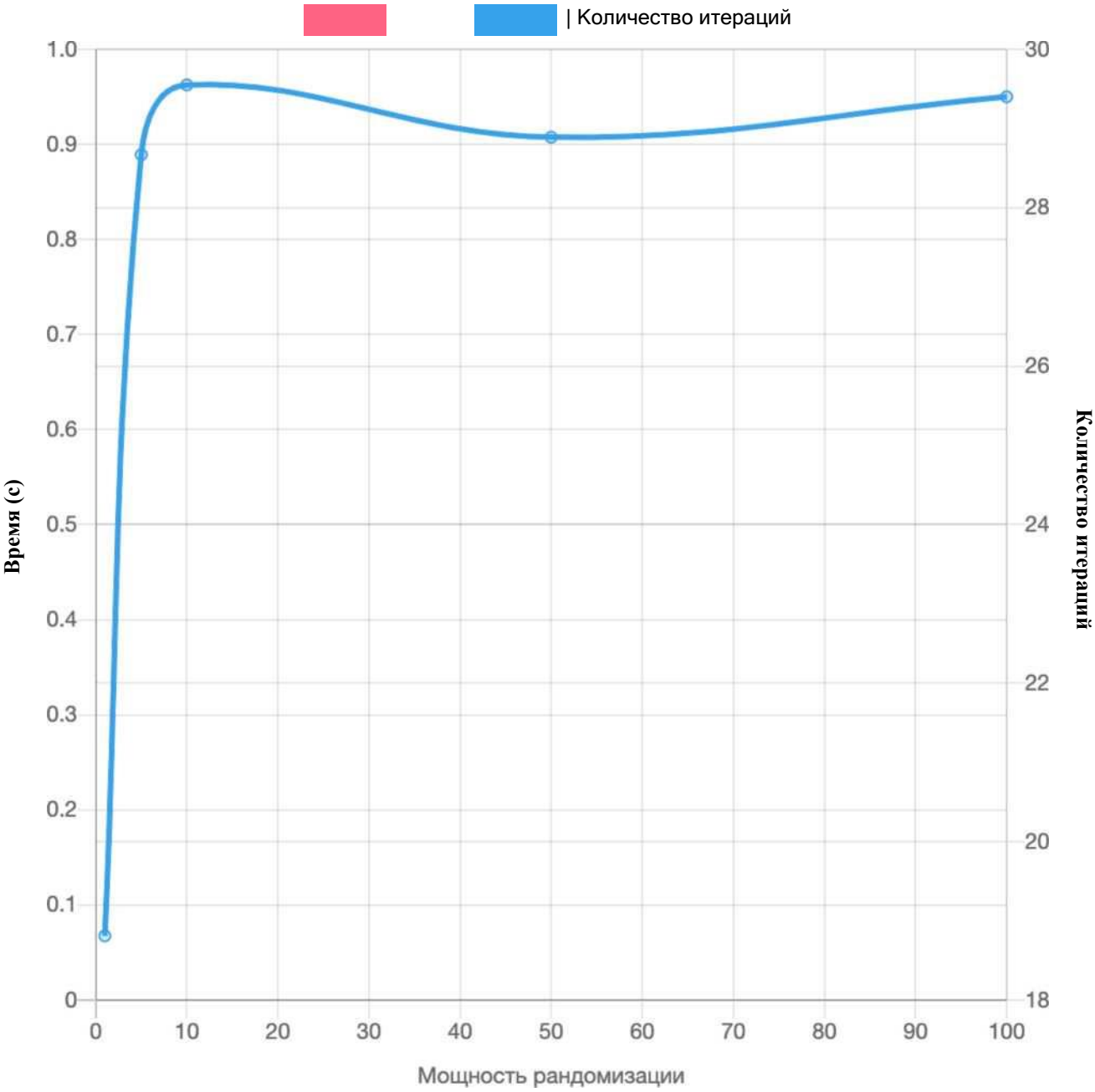

Рисунок 20 - Зависимость количества итераций алгоритма от мощности рандомизации при решении задачи размерности $n = 40$

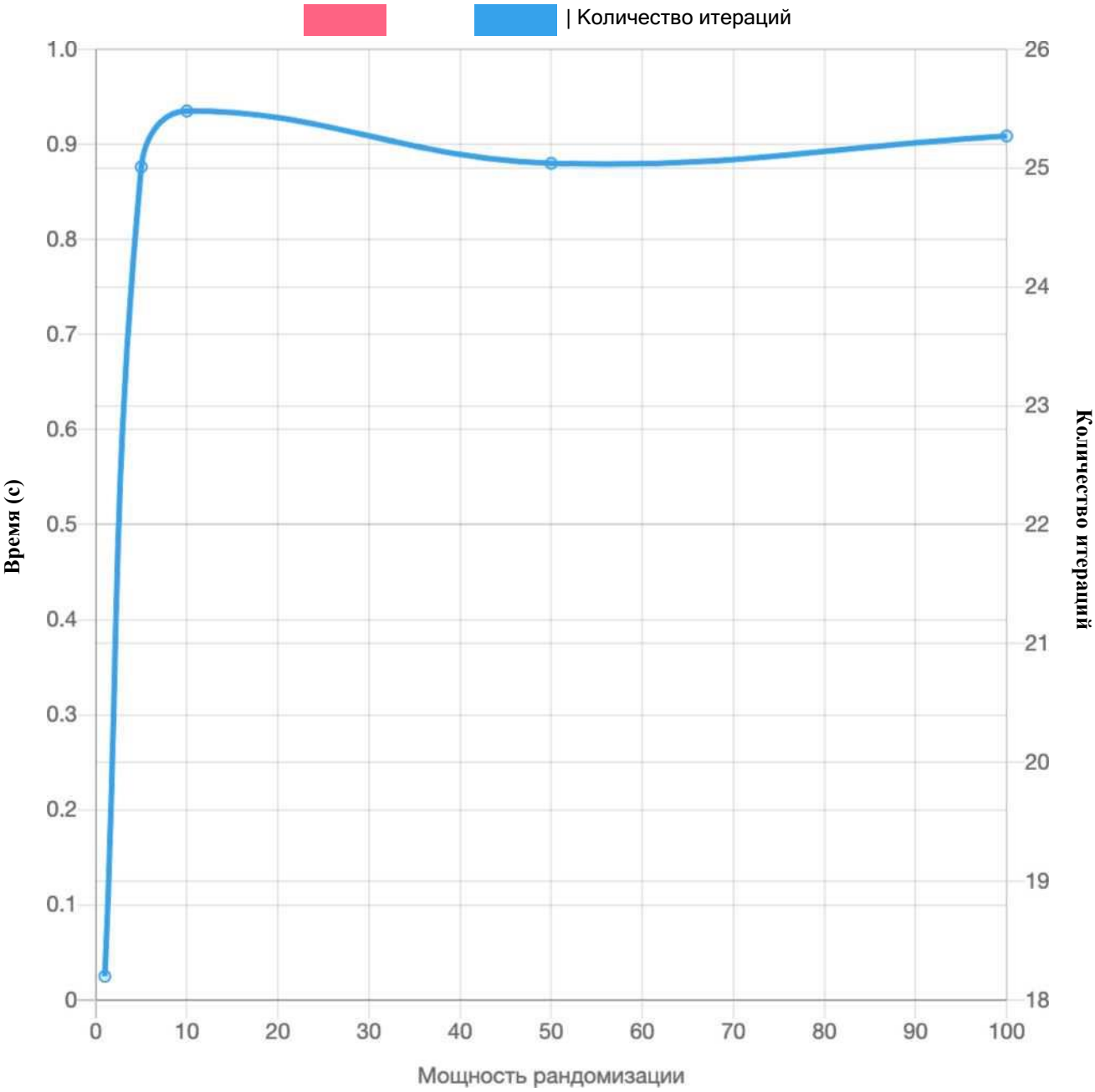

Рисунок 21 - Зависимость количества итераций алгоритма от мощности рандомизации при решении задачи размерности $n = 60$

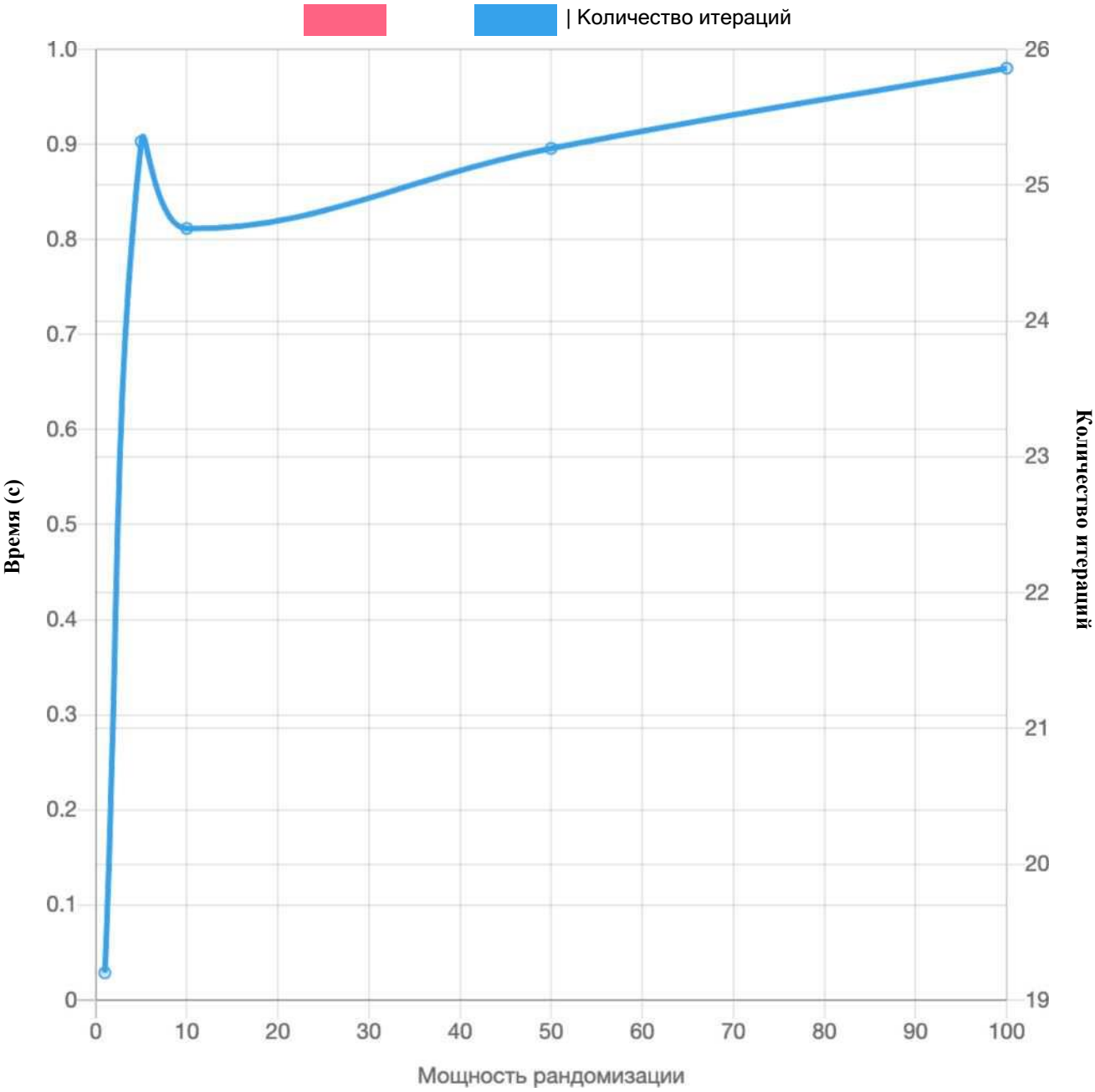

Рисунок 22 - Зависимость количества итераций алгоритма от мощности рандомизации при решении задачи размерности $n = 80$

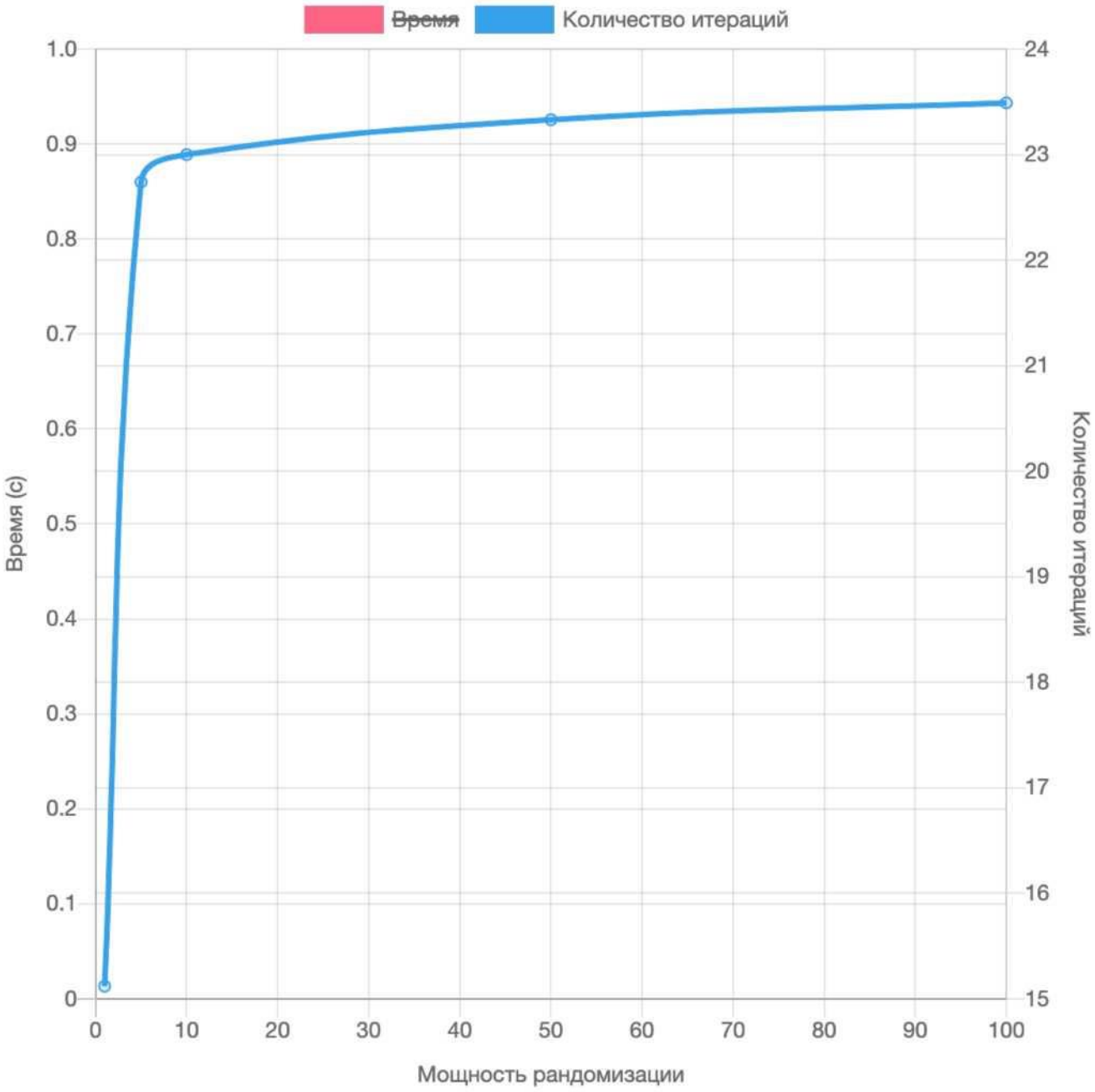

Рисунок 23 - Зависимость количества итераций алгоритма от мощности рандомизации при решении задачи размерности $n = 100$

Зависимости, представленные на рис. 19-23 носят однородный характер - при увеличении мощности рандомизации количество итераций алгоритма сначала резко увеличивается (на интервале $m < 10$), а затем стабилизируется (на интервале $10 < m < 100$).

ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы был исследован алгоритм решения задачи линейного программирования в условиях неполных данных.

В основе работы лежит анализ теоретических достижений науки в области решения задач линейного программирования и дискриминантного анализа, на базе которых и построен алгоритм.

После обзора предметной области были описаны непосредственно этапы алгоритма, а также представлена его программная реализация на базе клиентсерверного приложения на стеке технологий Vue.js (реализация клиентской части) и Node.js (реализация серверной части). Кроме того, был проведен ряд вычислительных экспериментов на предмет установления влияния параметров алгоритма на его быстродействие.

Направление дальнейших исследований может быть связано с оптимизацией кодовой базы и расширением функциональности как клиентской части приложения (добавление графической интерпретации решения), так и серверной (добавление новых эмуляторов модели эксперта).

БИБЛИОГРАФИЧЕСКИМ СПИСОК

1 Акулич, И.Л. Математическое программирование в примерах и задачах: Учебное пособие / И.Л. Акулич. - СПб.: Издательство «Лань», 2011. - 352 с.

2 Дискриминантный анализ [Электронный ресурс]. Режим доступа: shorturl.at/ijAH3. - (Дата обращения: 19.12.2018).

3 Мазуров, Вл.Д. Математические методы распознавания образов. Уч. пособие / Вл.Д. Мазуров. - 2-е изд., доп. и перераб. - Екатеринбург: Изд-во Урал, 2006. - 101 с.

4 Математические методы распознавания образов. Курс лекций [Электронный ресурс]. Режим доступа: shorturl.at/gjxT5. - (Дата обращения: 19.12.2018).

5 Распознавание образов [Электронный ресурс]. Режим доступа: shorturl.at/cuGNX. - (Дата обращения: 19.12.2018).

6 Математические методы распознавания образов [Электронный ресурс]. Режим доступа: shorturl.at/QSZ04. - (Дата обращения: 19.12.2018).

7 Математические методы в экономике / И.И. Еремин, Вл.Д. Мазуров, В.Д. Скарин, М.Ю. Хачай. - Екатеринбург: Изд-во «У-Фактория», 2000. - 280 с.

8 Еремин, И.И. Нестационарные процессы математического программирования / И.И. Еремин, В.Д. Мазуров. - М.: Наука. Главная редакция физико-математической литературы, 1979. - 287 с.

9 Гасс, С. Путешествие в страну линейного программирования / Сол Гасс; пер. с англ. Ю.Н. Сударева. - М.: «Мир», 1971. - 176 с.

10 Юдин, Д.Б. Линейное программирование (теория, методы и приложения) / Д.Ю. Юдин, Е.Г. Гольштейн. - М.: Наука. Главная редакция физико математической литературы, 1969. - 424 с.

11 Ашманов, С. А. Линейное программирование / С.А. Ашманов. - М.: Наука. Главная редакция физико-математической литературы, 1981. - 340 с.

12 Васильев, Ф.П. Линейное программирование / Ф.П. Васильев, А.Ю. Иваницкий. - М.: Изд-во «Факториал», 1998. - 176 с.

13 Барсов, А.С. Что такое линейное программирование / А.С. Барсов. - М.: Государственное издательство физико-математической литературы, 1959. - 104 с.

14 Данциг, Дж. Линейное программирование, его обобщения и применения / Джордж Данциг; пер. с англ. Г.Н. Андрианова, Л.И. Горькова, А.А. Корбута, А.Н. Ляпунова. - М.: Изд-во «Прогресс», 1966. - 590 с.

15 Карманов, В.Г. Математическое программирование: Учеб. пособие / В.Г. Карманов. - 5-е издание., стереотип. - М.: ФИЗМАТЛИТ, 2004. - 264 с.

16 Косоруков, О.А. Исследование операций: Учебник / О.А. Косоруков, А.В. Мищенко. - М.: Издательство «Экзамен», 2003. - 448 с.

17 Ромакин, М.И. Элементы линейной алгебры и линейного программирования / М.И. Ромакин. - М.: Государственное издательство «Высшая школа», 1963. - 273 с.

18 Гасс, С. Линейное программирование (методы и приложения) / Сол Гасс; пер. с англ. Е.Г. Гольштейна, М.И. Сушкевича. - М.: Государственное издательство физико-математической литературы, 1961. - 301 с.

19 Факторный, дискриминантный и кластерный анализ / Дж.-О. Ким, Ч.У. Мьюллер, У.Р. Клекка, М.С. Олдендерфер, Р.К. Блэшфилд. - М.: Финансы и статистика, 1989. - 215 с.

20 Вапник, В.Н. Теория распознавания образов (статистические проблемы обучения) / В.Н. Вапник, А.Я. Червоненкис. - М.: Наука. Главная редакция физикоматематической литературы, 1974. - 416 с.

21 Ту, Дж. Принципы распознавания образов / Дж. Ту, Рафаэль Гонсалес; пер. с англ. И.Б. Гуревича. - М.: «Мир», 1978. - 401 с.

22 Еремин, И.И. Введение в теорию линейного и выпуклого программирования / И.И. Еремин, Н.Н. Астафьев. - М.: Наука. Главная редакция физикоматематической литературы, 1976. - 190 с.

23 Лунгу, К.Н. Линейное программирование. Руководство к решению задач / К.Н. Лунгу. - М.: ФИЗМАТЛИТ, 2005. - 128 с.

24 Фу, К. Структурные методы в распознавании образов / К. Фу; пер. с англ. Н.В. Завалишина, С.В. Петрова, Р.Л. Шейнина. - М.: «Мир», 1977. - 312 с.

25 Соколинская, И.М. Синтез симплекс-метода и метода линейной коррекции в задачах линейной оптимизации с неформализованными ограничениями / И.М. Соколинская. - М.: Вычислительные методы и программирование, 2005. - 13 с.

26 Дуда, Р. Распознавание образов и анализ сцен / Ричард Дуда, Питер Харт; пер. с англ. Г.Г. Вайнштейна, А.М. Васьковского. - М.: «Мир», 1976. - 502 с.

27 Пантелеев, А.В. Методы оптимизации в примерах и задачах: Учеб. пособие / А.В. Пантелеев, Т.А. Летова. - 2-е изд., исправл. - М.: Высш. шк., 2005 - 544 с.

28 Патрик, Э. Основы теории распознавания образов / Эдвард Патрик; пер. с англ. Б.Р. Левина. - М.: Сов. радио, 1980. - 408 с.

29 Гренандер, У. Лекции по теории образов: Анализ образов / У. Гренандер; пер. с англ. И. Гуревича. - М.: «Мир», 1981. - 448 с.

30 Чабан, Л.Н. Теории и алгоритмы распознавания образов. Учебное пособие / Л.Н. Чабан. - М.: МИИГАиК, 2004. - 70 с.

31 Лепский, А.Е. Математические методы распознавания образов: Курс лекций / А.Е. Лепский, А.Г. Броневич. - Таганрог: Изд-во ТТИ ЮФУ, 2009. - 155 с.

32 Гренандер, У. Лекции по теории образов: Синтез образов. Учебное пособие / У. Гренандер; пер. с англ. И. Гуревича, Т. Дадашева. - М.: «Мир», 1979. - 376 с.

33 Нильсон, Н. Обучающиеся машины / Нильс Нильсон; пер. с англ. А.А. Дорофеюка. - М.: «Мир», 1967. - 180 с.

34 Муртаф, Б. Современное линейное программирование / Брюс Муртаф; пер. с англ. - М.: «Мир», 1984. - 224 с.

35 Еремин, И.И. Общая теория устойчивости в линейном программировании / И.И. Еремин. - Известия ВУЗов. Математика, 1999. - с. 43-52.

36 Гонсалес, Р. Цифровая обработка изображений / Рафаэль Гонсалес, Ричард Вудс. - М.: Техносфера, 2012. - 1104 с.

ПРИЛОЖЕНИЕ А

app.js

```
1: import express from 'express';
2: import cors from 'cors';
3: import multer from 'multer';
4: import { performance } from 'perf_hooks';
5:
6: import { expertise, check, get2LastPoints } from './utils';
7: import { get } from './_lincor';
8: import Parser from './parser';
9: import Solver from './solver';
10
11 const app = express();
12 app.use(cors());
13 const upload = multer({ dest: 'uploads/' });
14
15 app.post(
16     '/api/solve',
17     upload.single('model'),
18        async ({ body: { accuracy, radius, power }, file: {
   path } }, res) => {
19 try {
13              const parser = new Parser();
21              const {
22              model,
23              points,
24                  unformalized: TRUE_DIVIDED_FUNCTION,
25              } = await parser.parse(path);
26
27              const EXPERTISER_OPTIONS = {
28              randomization: {
29              radius,
30                      power: +power,
31                  },
32                  TRUE_DIVIDED_FUNCTION,
33              };
34
35
36                 const solver = new Solver(model);
37                 const size = solver.getSize();
```

```
38              const start = performance.now();
39
40              let iterations = 0;
41
42                          const iter = points => {
43                              iterations += 1;
44
45                const pointsCopy = points.map(point => ({
... point }));
46
47                  const { coeffs: fn, force } = get(size,
pointsCopy);
48
49              const solution =
solver.updateUnformalized(fn).solve();
50
51                if (force) {
52                    console.log('force');
53
54
55                      return { solution, points };
56                  }
57
58                      const canWeExit =
check(...get2LastPoints(pointsCopy), solution, accuracy);
59
60                      if (!canWeExit) {
                    return iter(expertise(points, solution,
EXPERTISER_OPTIONS));
61              } else {
62                  return { solution, points };
63              }
64          };
65
66          const solution = iter(points);
67
68          const stop = performance.now();
69
70          res.status(200).send({
71              solution,
72              time: { x: +power, y: (stop - start) / 1000
73 },
:                   iterations: { x: +power, y: iterations },
74                  key: +power
·
```

```
75:             });
76:         } catch ({ message }) {
77:             res.status(500).send({ status: 500, message });
78:         }
79:     }
80: );
81:
82:  app.listen(3000, () => console.log('Example app listening
   on port 3000!'));
```

lincor.js

```
          DOUBLE_ONE, DOUBLE_ZERO, mul2DoublePoints
   './utils';
2: import { performance } from
3:
4: export const get = (size, points) => {
       let coeffs = Array (size + 1).Fill()OUBLE_ZERC );
6:    let canWeExit = false;
7:
8:    // save concat operation
9:    // recursion / iterative
10:     // math.js
11:     const cache = {};
12:
13:     const start = performance.now();
14:     let stop = null;
15:
16:     while (!canWeExit)  {
17:          // exit if flag and value not equal
18:          const newPoints = points.map(({ x, flag }) => {
19:              const extraPoint = x.concat(DOUBLE_ONE);
20:
21:              const phiValue = mul2DoublePoints(coeffs,
   extraPoint);
22
23:              if (flag === 1) {
24                   if (phiValue..e()OUBLE_ZERC)) {
25:                      coeffs = coeffs.map((coeff, index) =>
  coeff.add(extraPoint[index]));
26:                  }
27:              } else {
28                   if ( phiValue.       ge       (
29                   )OUBLE_ZERC)) {                 index) =>
  :                     coeffs = coeffs.map((coeff,
30coeff.sub(extraPoint[index]));
31:                  }
32:              }
33
34:              return {
  :                   x.
```

```
35            flag,
36            value: phiValue,
37            };
38            });
39
40            stop = performance.now();
41
42            canWeExit =
43            !newPoints.some(
44            ({ flag, value }) =>
45                        (flag === 1 && value.le(DOUBLE_ZERO))


                        (flag === -1 && value.ge(DOUBLE_ZERO))
47:               ) || (stop - start) / 1000 > 60;
48:       }
49:
50:       const force = (stop - start) / 1000 > 60;
51:
52:       return { coeffs: coeffs.map(num => num.toNumber()),
  force };
53:   };
```

ПРИЛОЖЕНИЕ В

utils.js

```
54   import { readFile } from 'fs';
55   import Double from 'double.js';
56
57   export const DOUBLE_ZERO = new Double('
58   export const DOUBLE_ONE = new Double('1');
59   export const DOUBLE_MINUS_ONE = new Double('
54   // export const DOUBLE_TWO = new Double('2');
61
62   export const rfa = path => {
63     return new Promise((reso£ve, reject) => {
64     readFile(path, 'utf8', (err, data) => {
65     if (err) {
66     reject();
67             }
68
69             resolve(data);
70         });
71     });
72   };
73
74 : export const distanceBetweenPoints = (pointl, point2) => {
75 :     const distance = point2.x.reduce(
76 :         (distance, coord, index) =>
   distance.add(coord.sub(point1.x[index]).sqr()),
77:         DOUBLE_ZERO
78:     );
79:
        if (distance.eq(DOUBLE_ZERO)) {
            return DOUBLE_ZERO;
82:     } else  {
83:         return distance.sqrt();
84:     }
85: };
86:
87: export const get2LastPoints = points => {
88:     const length  =points.length;
89:
90:     return[points[length   - 1], points[length - 2]];
```

```
export const check = (LastPoint, preLastPoint, solution, accuracy) =>
    distanceBetweenPoints(lastPoint, solution)
        .add(distanceBetweenPoints(preLastPoint, lastPoint))
        .lt(new Double(accuracy));

export const mul2DoublePoints = (pointl, point2) =>
    point1.reduce(
        (sum, coeff, index) => coeff.mul(point2[index]).add(sum),
        DOUBLE_ZERO
    );

export const duplicate = (points, target) =>
    points.some(({ x }) => x.every((coord, index) => coord.eq(target.x[index])));

export const random = (coord, radius) => {
    const doubleRadius = new Double('${radius}');
    const doubleRandom = new Double('${Math.random()}');

    const newCoord = doubleRadius.mul(doubleRandom);

    return coord.add(newCoord);
    // return coord.add(doubleRadius);
};

export const expertise = (points, solution, options) => {
    const {
        randomization: { radius, power },
        TRUE_DIVIDED_FUNCTION,
    } = options;

    let solutions = [];

    if (duplicate(points, solution)) {
        solutions = Array(power)
            .fill()
```

```
130:         solutions = [solution];
131:     }
132:
133:     return points.concat(
134:         solutions.map(({ x }) => {
135:             const extraSolution =
  x.concat(DOUBLE_MINUS_ONE);
136:
137:             return {
138:                 x,
139:                 flag:
  mul2DoublePoints(TRUE_DIVIDED_FUNCTION,
  extraSolution).gt(DOUBLE_ZERO)
140:                     ? 1
141:                     : -1,
142:                 value: DOUBLE_ZERO,
143:             };
144:         })
145:     );
146: };
147:
148: export const generate = (points, target, radius) => {
149:     const targetCopy = { ...target };
150:     // radius = getRadius(points);
151:
152:     console.log('radius', radius);
153:     console.log('random');
154:
155:     while (duplicate(points, targetCopy)) {
156:         targetCopy.x = targetCopy.x.map(coord =>
  random(coord, radius));
157:     }
158:
159:     return targetCopy;
160: };
161:
162: export const getRadius = points => {
163:     const pointsCopy = points.map(point => ({ ...point }));
164:
165:     const reversePointsCopy = pointsCopy.reverse();
166:
167:     // check that points are  really closer
```

```
168:     const A = reversePointsCopy.find(point => point.flag
   === 1);
169:     const B = reversePointsCopy.find(point => point.flag
   === -1);
170:
171:     const distance = distanceBetweenPoints(A, B);
172:
173:     return distance.toNumber() / 2;
174: };
```

parser.js

```
175: import Double from 'double.js';
176:
177: import { rfa, DOUBLE_ZERO } from './utils';
178:
179: export default class Parser {
180:     // refactor
181:     _parse({ objective, constraints, unformalized, A, B }) {
182:         const variables = objective.reduce(
183:             (variables, objectiveCoeff, objectiveCoeffIndex) => {
184:                 return {
185:                     ...variables,
186:                     ['x${objectiveCoeffIndex + 1}']: {
187:                         capacity: objectiveCoeff,
188:                         phi: unformalized[objectiveCoeffIndex],
189:                         ...constraints.reduce(
190:                             (constraintsCoeffs, constraintCoeff, constraintCoeffIndex) => {
191                                 return {
192                                     ...constraintsCoeffs,
193                                     ['constraint${constraintCoeffIndex + 1}']: constraintCoeff[
194                                         objectiveCoeffIndex
195                                     ],
196                                 };
197                             },
198                             {}
199                         ),
194                     },
201                 };
202             },
203             {}
204         );
205
203         return {
207             model: {
```

```
208                 optimize: 'capacity',
209                 opType: 'max',
208                 constraints: {
211                     ...constraints.reduce(
212                         (constraints, constraint, index) =>
   ({
213:                            ...constraints,
214:                            ['constraint${index + 1}']: {
215:                            max:
   constraint[constraint.length -1],
216:                            },
217:                        }),
218:                        {}
219:                    ),
220:                    phi: {
221:                        max:
   unformalized[unformalized.length - 1],
222                     },
223                 },
224                 variables,
225                 objective,
226             },
227             points: [...A,    .B],
228             unformalized,
229         };
230     }
231
232     async parse(path) {
233         const data = await rfa(path);
234
235         const model =
   [...data.matdiAll(/\[(.*)\]\r?\n([A[]+)/g)].reduce(
236             (model, [, key, vaLue]) => ({
237                 ...model,
238                 [key]: this.parseLine(key, value.trim()),
239             }),
240             {}
241         );
242
243         if (Object.values(model).some(eLement => element
   === undefined)) {
244:             throw new Error('Error is occured while file
   reading!');
```

```
245          }
246
247          return this._parse(model);
248      }
249
250                  parseLine(fcey, value) {
251                       switch (key) {
252        case 'objective':
253            return value.split(';').map(coeff =>
  +coeff.trim());
254:                 case 'unformalized':
255:                     return value
256:             .replace('<=', ';')
257:             .split(';')
258:             .map(coeff => new
  Double(coeff.trim()));
259                             case 'constraints':
259             return value.split(Ar?\n/).map(1ine => {
261                 // <= используется для удобочитаемости
262                 return line
263             .replace('<=', ';')
264             .split(';')
265             .map(coeff => +coeff.trim());
266             });
267             case 'A':
268             case 'B':
269                                 return value
270             .split('\n')
271             .map(1ine => line.trim())
272             .map(1ine => ({
273             x: line.split(';').map(number =>
                   new Double(number.trim())),
274                          flag: key === 'A' ? 1 : -1,
:                          value: DOUBLE_ZERO,
275                      }));
:            }
276      }
:
```

solver.js

```
280 import solver from 'javascript-lp-solver';
281 import Double from 'double.js';
282
283 export default class Solver {
284 constructor(model) {
285 this.model = model;
286 }
287
288 getSize() {
289     return this.getObjective().length;
290 }
291
292 getObjective() {
293     return this.model.objective;
294 }
295
296 getConstraints() {
297     return this.model.constraints;
298 }
299
300 updateUnformalized(coe//s) {
301     coeffs.forEach((coeff, index) => {
302         if (index === coeffs.length - 1) {
303             this.model.constraints.phi.max = -coeff;
304         } else {
305             this.model.variables['x${index + 1}'].phi =
  coeff;
306:             }
307:         });
308:
309:          return this;
310:     }
311:
312:     solve() {
313:          const solution = solver.Solve(this.model);
314:
315:          return {
316:              x: Array(this.getSize())
```

```
317                 .fillO
318                 .reduce(
319                 (x, _, index) => [...x, new
  Double('${solution['x${index + 1}']}')],
320                     []
321                 ),
322         };
323     }
324 }
325
326 export default class Solver {
327 constructor(model) {
328 this.model = model;
329 }
330
331 getSize() {
332 return this.getObjective().length;
333 }
334
335 getObjective() {
336 return this.model.objective;
337 }
338
339 getConstraints() {
340 return this.model.constraints;
341 }
342
343 updateUnformalized(coe//s) {
344 coeffs.forEach((coeff, index) => {
345     if (index === coeffs.length - 1) {
346         this.model.constraints.phi.max = -coeff;
347     } else {
348         this.model.variables['x${index + 1}'].phi =
  coeff
349:             }
350:         });
351:
352:         return this;
353:     }
354:
355:     solve() {
356:         const solution = solver.Solve(this.model);
357:
```

```
358:         return {
359:             x: Array(this.getSize())
360:                 .fill()
361:                 .reduce(
362:                     (x, _, index) => [...x, new
  Double('${solution['x${index + 1}']}')],
363:                     []
364:                 ),
365:         };
366:     }
367: }
368:
```