САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА МАТЕМАТИЧЕСКОЙ ТЕОРИИ ИГР И СТАТИСТИЧЕСКИХ РЕШЕНИЙ

Марченко Александра Андреевна

Выпускная квалификационная работа магистра

Разработка методологических инструментов повышения экономической привлекательности транспортного коридора «Северный морской путь»

Направление:
«Исследование операций и системный анализ ВМ.5504.2016»

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Крылатов А. Ю.

Санкт-Петербург
2018

Оглавление

Введение

Огромное значение в системе транспортного обеспечения Арктики имеет Северный морской путь (СМП). Севморпуть входит в состав почти 70% пространства территории Российской Федерации (Рис.1). Возможность прохода Северным морским путем впервые обосновал великий русский ученый М.В. Ломоносов, обобщив результаты экспедиций полярных исследователей России.

Рисунок 1. Северный морской путь - транспортная артерия страны.

Потенциал Арктической зоны заключается в развитии логистической магистрали, как для внутренних перевозок, так и для транзитных. В последнее время навигация стала более продолжительной из-за таяния полярных льдов и это способствует проработке политики тарифного образования для прохождения пути, модернизации ледокольного флота для сопровождения, а так же

модернизации инфраструктур стыка логистических узлов с помощью мониторинга, моделирования и координация движения судов.

Путин В.В. отмечает: «СМП позиционируется как международная транспортная артерия, способная составить конкуренцию традиционным морским трассам, как по стоимости перевозок, так и по безопасности транспортного сообщения. Севморпуть на треть короче традиционного южного маршрута, что дает возможность российским и иностранным перевозчикам существенно сократить транспортные расходы и получить весомые экономические преимущества» [1, 88; 2,1].

Много статей и публикаций про экономичное использование СМП как путь между Европой и Азией, так как он самый короткий. При этом все показатели найдены путем расчета простых математических манипуляций. Поэтому, целью данной работы является поиск методологического решения распределения транспортных потоков с учетом реального спроса на рынке морских транспортных грузоперевозок, на основе которых можно строить конкурентно способную политику тарифного образования и прогнозирование потока для модернизации портов.

Постановка цели и задачи

Целью данной работы является разработка математической модели распределения транспортных потоков с использованием Северного морского пути в качестве национальной единой транспортной коммуникации Российской Федерации в Арктике.

Данная работа предполагает обработку большого массива материалов и значительную трудоемкость для достижения высокой степени детализации.

Для успешного выполнения поставленной цели было выделено пять блоков задач:

1. Изучение моделей, применяемых для распределения транспортных потоков.
2. Исследование сети международных морских грузопотоков. Выявление условий экономической целесообразности работы Северного морского пути.
3. Сбор данных, проведение анализа и подбора, наиболее подходящих к реальному времени, основных характеристик грузопоточных мощностей международных транзитов.
4. Калибровка разработанной модели на основе собранных данных.
5. Предоставление методических рекомендаций для повышения экономической привлекательности транспортного коридора.

Обзор литературы

В данной работе были использованы данные с научных журналов, отдельных статей и издательств, располагаемых доступной информацией в виде таблиц, диаграмм и графиков.

Для начала работы был изучен обзор основных методов и идей в области математического моделирования транспортных потоков. Изучив публикацию «Математическое моделирование транспортных потоков» Швецова В.И. [3, 46], составила общее представление о своей будущей работе. А именно, как и откуда отталкиваться, как подходить к изучению, как применять классические транспортные модели и пробовать их модернизировать для конкретной задачи.

В результате был сделан вывод: рассмотреть и попробовать разработать две модели распределения потоков морских судов с использованием альтернативного СМП: с фиксированным спросом и с эластичным спросом. Модели распределения потока в данном случае, являются оптимизационными моделями с дескриптивным подходом. При этом подходе моделирования выносится предположение, что структура транспортных потоков и их распределение формируется исходя из индивидуальных решений каждого участника движения. Они могут быть основаны на оптимизации времени, затрат или других личных критериев.

Основной теоретической базой для выпускной работы послужила статья научного руководителя Крылатова А.Ю. «Оптимальные стратегии управления транспортными потоками на сети из параллельных каналов» [4, 127]. Наряду с другими работами, написанными в соавторстве с Захаровым В.В. и Шиховой К.А. [5, 6, 7] появилась основа для разработки модели с эластичным спросом. Так же были приняты, так называемые, принципы Wardrop J. G. [8, 325] для равновесного распределения транспортных потоков. Эти принципы являются фундаментальными для многих классических транспортных моделей. Его математическая формулировка способна упростить расчеты в зависимости от условий поставленной задачи.

Для расчетных данных найти информацию было куда сложнее, так как точные цифры по пропускной способности того или иного пути, а так же стоимость перехода и ледокольного сопровождения, документально не зафиксированы. Многие данные либо отсутствуют, либо вычислены аналитическим и статистическим анализом внешнего рынка спроса на морские грузоперевозки.

Некоторые показатели были взяты из работы Лукина Ю.Ф. «Анализ деятельности Северного морского пути» [9, 469]. По истечению времени, для более реального применения моделей, показатели были усреднены, так как на разных источниках использовались разные величины тех или иных факторов.

Ключевые экономические и инженерные комментарии к данной работе были взяты из конференции на «Международном молодежном форуме Арктика. Сделано в России», на пленарном заседании IV Международного арктического форума «Арктика – территория диалога» и на докладе ведущего сотрудника с Института Проблем Транспорта Российской Академии Наук (ИПТ РАН), к.т.н. Д.В. Козьмовского на тему «Проблемы развития Северного морского пути как международного транзитного коридора».

Глава 1. Распределение потоков морских судов с фиксированным спросом

§1.1. Моделирование

Рассматриваем транспортную сеть с одной парой Исток-Сток и n параллельных маршрутов между ними. Исток является пунктом отправления, а сток пунктом прибытия соответственно. Будем иметь в виду, используя принцип Вардропа [8, 325], что все участники движения выбирают маршрут, при котором индивидуальные затраты будут минимальными.

Введем обозначения:

- i - номер рассматриваемого маршрута, $i \in \{1, n\}$;
- $F > 0$-общее количество участников движения по всем маршрутам из истока в сток, корабли;
- $f_i \geq 0$ - объем морских судов, направленных по i-му маршруту;
- $f = (f_1, \dots, f_n)$ - вектор распределения судов по n маршрутам;
- $f^* = (f_1^*, \dots, f_n^*)$-оптимальный вектор распределения судов по n маршрутам, среди всех рассматриваемых векторов;
- t_i^0 - время чистого пути по i-му маршруту;
- $c_i > 0$- пропускная способность i-го маршрута;
- $d_i(f_i) > 0$ - функция задержки потока f_i на маршруте i;
- kue – маршрут с наибольшим временем прохождения пути среди всех рассматриваемых маршрутов;
- t^*-оптимальное время перехода из Истока в Сток по всем рассматриваемым маршрутам.

Используя обозначения, рассмотрим функцию задержки, которая предложена в методике расчетах Бюро общественных дорог в США:

$$d_i(f_i) = t_i^0 \left(1 + \frac{f_i}{c_i}\right), \qquad \forall i \in \{1, n\} \tag{1}$$

Введем определение конкурентного равновесия в сети по принципу Вардропа [8, 325]. Распределение потока участников движения *F* по рассматриваемым дугам сети происходит таким образом, что затраты на переход из Истока в Сток по другим не используемым маршрутам превышают затраты на передвижение в рассматриваемых дугах.

Математическая интерпретация конкурентного равновесия описывается, как:

$$d_i(f_i) \begin{cases} = t^*, \text{при } f_i > 0, \\ > t^*, \text{при} f_i = 0, \end{cases} \qquad \forall i \in \{1, n\} \tag{2}$$

Решением данной главы и модели состоит в нахождении вектора распределения судов $f = (f_1, \dots, f_n)$ по маршрутам, для которого будут выполнены условия не отрицательности аддитивного потока и принципа Вардропа.

С учетом этого задача выглядит следующим образом:

$$Z(f^*) = \min_f \left\{ \sum_{i=1}^n \int_0^{f_i} t_i^0 \left(1 + \frac{u}{c_i}\right) du \right\}, \tag{3}$$

$$\sum_{i=1}^n f_i = F\ , \tag{4}$$

$$f_i \geq 0, \qquad \forall i \in \{1, n\} \tag{5}$$

В статье Крылатова А.Ю. «Оптимальные стратегии управления транспортными потоками на сети из параллельных каналов» [4, 127] приведено доказательство, которое показывает что при данной формулировке функции цели (3), можно прийти к ситуации, в которой каждый участник движения минимизирует свои личные затраты. Таким образом, выписав функцию Лагранжа уравнения (3) и продифференцировав, получим функцию задержки потока (1) при $f_i > 0, \forall i \in \{1, n\}$.

Если мы найдем оптимальное распределение, то затраты по всем альтернативным путям будут одинаковы. Если кто-то поменяет свою стратегию:

уйдя с одного маршрута, уменьшит затраты остальных участников, а на тот на который перейдет-увеличит затраты. Поэтому это состояние является равновесным.

Для удобства предстоящих расчетов пронумеруем маршруты по увеличению времени прохождения чистого пути

$$t_1^0 \le t_2^0 \le \cdots \le t_n^0 \tag{6}$$

Используя выражение (3) с ограничениями (4) и (5), получаем задачу оптимизации, решение, которого, приводит транспортный поток к конкурентному равновесию. Справедлива следующая

Теорема 1. При реализации распределения транспортного потока

$$f_i = \begin{cases} \frac{c_i}{t_i^0}\frac{F+\sum_{i=1}^{kue} c_i}{\sum_{i=1}^{kue}\frac{c_i}{t_i^0}} - c_i, \text{при } i \le kue, \\ 0, \text{ при } i > kue \end{cases} \tag{7}$$

задача (3) - (5) при выполнении (6) достигает конкурентного равновесия.

Где F находится исходя из

$$F > \sum_{i=1}^{kue} c_i \left(\frac{t_{kue}^0}{t_i^0} - 1\right) \tag{8}$$

По дальнейшим расчетам будет видно, что если строгое неравенство изменится, и спрос будет равен или меньше правой части уравнения, то по пути *kue* никто не поедет и распределение, в данном случае на этом пути, будет равно 0 (объем перевозок распределится между остальными маршрутами).

Доказательство:

Введем такое $kue \in \{1, n\}$, что при выполнении (6) $t_{kue}^0 < d_i\left(f_i\right)$, при этом $t_{kue+1}^0 \ge d_i\left(f_i\right)$.

Тогда подставив (4) в

$$f_i = \begin{cases} \left(\frac{d_i(f_i)}{t_i^0} - 1\right) c_i, \text{при } t_i^0 < d_i(f_i), \\ 0, \text{ при } t_i^0 \ge d_i(f_i), \end{cases} \qquad \forall i \in \{1, n\} \tag{9}$$

Получим

$$\sum_{i=1}^{n} f_i = \sum_{i=1}^{kue} \left(\frac{d_i(f_i)}{t_i^0} - 1 \right) c_i = F \quad (10)$$

Таким образом, отсюда

$$d_i(f_i) = \frac{F + \sum_{i=1}^{kue} c_i}{\sum_{i=1}^{kue} \frac{c_i}{t_i^0}} \quad (11)$$

После, подставив (11) в (9), придем к (7).

Теорема доказана.

§1.2. Эффективность модели на реальных данных при фиксированном спросе

Во многих источниках говорится о разных условиях, ценах и данных по передвижению выбранных мною маршрутов [9]. Реальные параметры собраны мною лично и используются по оптимальному экспертному выбору.

Данные могут меняться, в зависимости от текущей статистики или же от природных условий. В данной работе рассматриваются показатели, усредненные с максимально хорошими природными условиями.

Итак, рассматриваем непосредственно пути, в которых будет находиться вариант передвижения по Северному Морскому Пути (СМП). Выбрана Норвегия (порт Киркенес) - как Исток и Япония (Порт Иокогама) - как Сток.

Так же рассчитываем модель с альтернативными маршрутами прохождения через Суэцкий канал (Рис. 2) и маршрут, огибая Африку (Рис. 3). Отсюда, $n=3$.

Рисунок 2. Маршрут из Киркенес в Иокогаму через Суэцкий канал.

Рисунок 3. Маршрут из Киркенес в Иокогаму, огибая Африку.

По карте построенной навигацией морских путей онлайн, видно, что через СМП (Рис.4) проходит наименьшее количество морских судов, а по Суэцкому каналу идет переизбыток.

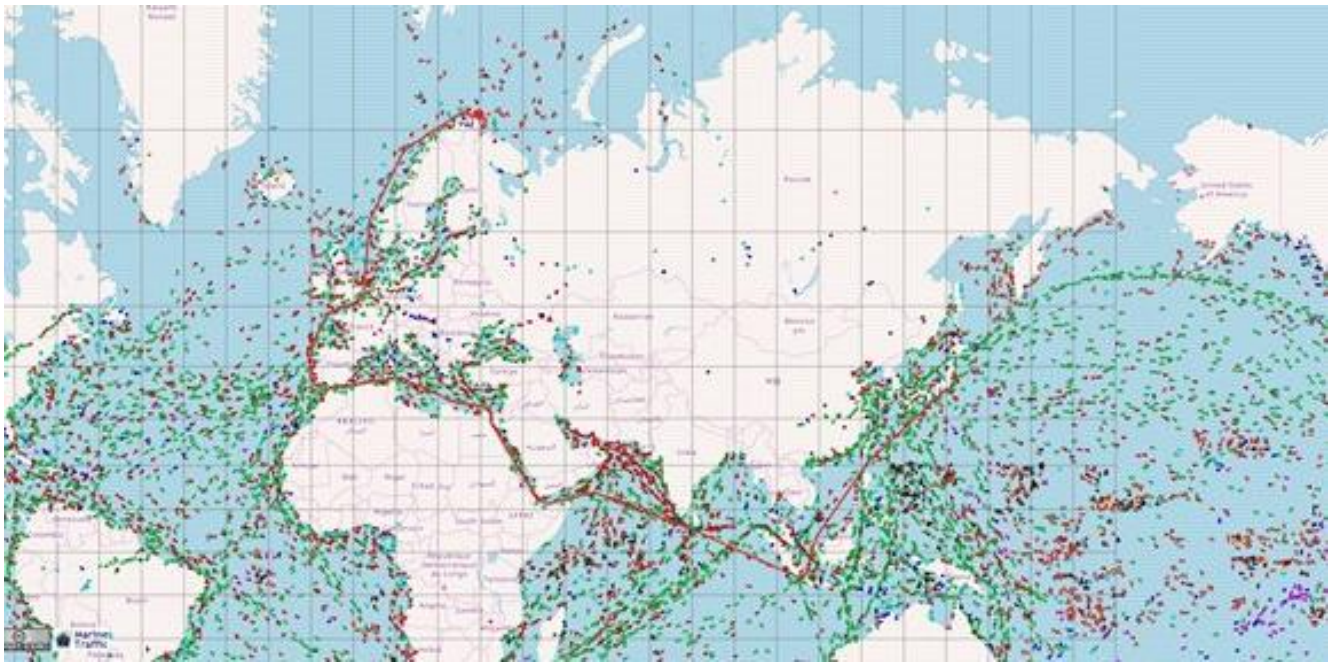

Рисунок 4. Онлайн навигатор движения морских судов и плотность распределения на морской поверхности.

Рассмотрим таблицу входных данных для модели (Таблица 1):

	СМП	СУЭЦ	АФРИКА
	i=1	i=2	i=3
Расстояние, км	≈9110 км	≈23000 км	≈28000 км
Время перехода, t_i	≈30 дней	≈48 дней	≈65 дней
Пропускная способность, c_i	50 млн. тонн в год	998 млн. тонн в год	86 млн. тонн в год
	321 корабль/месяц	1666 кораблей/месяц	364 кораблей/месяц
Чистое время пути, t_i^0	10 дней	34 дня	60 дней
	0,33	1,13	2

Таблица 1. Входные данные для расчета модели.

Здесь чистое время в пути взято из расчета кол-во дней/30 дней в месяце. Этот показатель показывает наиболее короткое время прохождения пути с наилучшими климатическими условиями и «пустой дороги».

Пропускная способность рассчитывалась, исходя из показателей прошлых грузоперевозок (тонн за год), примерного количества прохождения кораблей в год

и наличия ледоколов с возможным караваном из суден за ним. Она может меняться в зависимости от вместимости кораблей в порту, свободных ледоколов и политических среды.

Учитывая условие (6) пронумеруем маршруты по возрастанию времени прохождения чистого пути:

$$t_1^0 \leq t_2^0 \leq t_3^0$$

или

$$0{,}33 \leq 1{,}13 \leq 2$$

Видно, что из трех предложенных вариантов маршрута, наибольший по времени чистого пути является маршрут, который огибает Африку. С учетом этого, *kue* маршрут под номером 3 (i=3).

Исходя из показателей в таблице, найдем спрос на переход между Истоком и Стоком по формуле (8):

$$F \geq 321\left(\frac{2}{0{,}3} - 1\right) + 1666\left(\frac{2}{1{,}13} - 1\right) + 364\left(\frac{2}{2} - 1\right)$$

$$F \geq 3101{,}66$$

Если мы возьмем F = 3101,66 кораблей, то по дальнейшим расчетам будет видно, что распределение по маршруту 3 будет нулевым, так как (7) формула дает понять, что по этому пути нам не выгодно идти и затраты будут высоки. Если же мы возьмем F больше найденного числа, то по маршруту 3, огибая Африку, поплывут корабли и распределение с учетом затрат будет найдено с этим маршрутом.

По ряду экономических и политических причин, рассматривать альтернативой путь через Африку не будем, поэтому: F = 3101,66.

Найдем распределения для маршрута *i=1*, через СМП по (7) формуле:

$$f_1 = \frac{321}{0{,}3} \cdot \frac{3101{,}66 + 321 + 1666 + 364}{\frac{321}{0{,}3} + \frac{1666}{1{,}13} + \frac{364}{2}} - 321 = 1819$$

Найдем распределения для маршрута *i=2*, через Суэцкий канал:

$$f_2 = \frac{1666}{1{,}13} \cdot \frac{3101{,}66 + 321 + 1666 + 364}{\frac{321}{0{,}3} + \frac{1666}{1{,}13} + \frac{364}{2}} - 1666 = 1282{,}66$$

Найдем распределения для маршрута i=3, огибая Африку:

$$f_3 = \frac{364}{2} \cdot \frac{3101{,}66 + 321 + 1666 + 364}{\frac{321}{0{,}3} + \frac{1666}{1{,}13} + \frac{364}{2}} - 364 = 0$$

По расчетам видно, что условия модели (4) и (5) выполнены. Найдем функции задержек распределенных потоков по формуле (1):

$$d_1(f_1) = 0{,}3(1 + \frac{1819}{321}) = 2$$

$$d_2(f_2) = 1{,}13(1 + \frac{1282{,}66}{1666}) = 2$$

Отсюда, подставив значения функции задержки в (9) - (11) получим подтверждение доказательства Th 1.

Итак, используя (2), получим оптимальное время перехода

$$d_i(f_i)\begin{cases} = 2, \text{ при } f_i > 0, \\ > 2, \text{ при} f_i = 0, \end{cases} \qquad \forall i \in \{1{,}3\}$$

При всех остальных вариациях подбора спроса F, увеличивая его, мы увеличиваем значение функции задержки, а значит и функцию цели.

При таком распределении потока можно перейти из морского порта Норвегии в морской порт Японии за равновесное время (2 месяца), и это значит, что Северный морской путь будет актуальным маршрутом на сети:

$$f = (1819; 1282{,}66;\ 0)$$

В итоге, найдено оптимальное распределение, затраты по всем альтернативным путям одинаковы, по не альтернативным путям равны 0. Если кто-то поменяет свою стратегию: уйдя с одного маршрута, уменьшит затраты остальных участников, а на тот на который перейдет - увеличит затраты.

Для удобства сравнения с итогами следующей главы, предоставим результаты модели с фиксированным спросом (Таблица 2):

$\boldsymbol{\kappa}$	2
$\boldsymbol{t}^*$	2
$\boldsymbol{F}$	3101
$\boldsymbol{f}$	$(1819; 1282;\ 0)$

Таблица 2. Результаты модели с фиксированным спросом

Зная, что у маршрута $i=3$ чистое время передвижения равно 2 месяца, возникает вопрос, почему он не вошел при распределении, если оптимальное время тоже равно 2? Оптимальное время при такой загрузке будет равно двум месяцам, а в случае с Африкой, не имея никаких кораблей (никто не поехал) уже время равно 2.

Глава 2. Распределение потоков морских судов с эластичным спросом

§2.1. Моделирование

Рассматриваем ту же транспортную сеть с ее свойствами.

Если t*- оптимальное время прохождения из Истока в Сток по любому рассматриваемому маршруту входящего в данную модель с учетом всех ограничений, тогда $F = g(t^*)$ является функцией этого оптимального времени. Мы предполагаем, что спрос эластичен. Это значит, что F не фиксирован, а зависит от времени, которого придется ждать в случае затора сети.

Спрос не будет увеличиваться, если нам, допустим, от Парка Победы до центра города надо ехать 2 часа. Тогда человек не сядет в машину, а пойдет в метро. То есть существует какой-то предел, когда участник движения не выберет данный маршрут, так как происходит задержка движения на нём.

Соответственно,

$$t^* = g^{-1}(F)$$

Пусть в нашей модели существует такой T, который дает оценку того, сколько участники движения готовы потратить времени на любой маршрут из Истока в Сток.

Тогда

$$t^* = g^{-1}(F) = T - rF \tag{12}$$

, где r коэффициент, который изначально задан

Очевидно, что с увеличением времени, которого нам придется потратить, спрос падает.

Для упрощения дальнейшей работы преобразуем функцию задержки потока:

$$d_i(f_i) = t_i^0 \left(1 + \frac{f_i}{c_i}\right), \qquad \forall i \in \{1, n\}$$

$$d_i(f_i) = a_i + b_i f_i , \qquad \forall i \in \{1, n\} \tag{13}$$

, очевидно, что

$$a = t_i^0 \tag{14}$$

$$b = \frac{t_i^0}{c_i} \tag{15}$$

Так же у нас остаются условия (2), (4) и (5).

Решение данной главы и модели состоит в нахождении вектора распределения судов $f = (f_1, \dots, f_n)$ по маршрутам с учетом эластичности спроса F от оптимального времени передвижения t*.

В итоге в предложенных обозначениях (12) – (15) математическая задача (3) – (5) преобразуется таким образом:

$$Z(f^*, F^*) = \min_{f.F} \left\{ \sum_{i=1}^{n} \int_0^{f_i} d_i(u)du - \int_0^{F} g^{-1}(u)du \right\}, \tag{16}$$

$$\sum_{i=1}^{n} f_i = F , \tag{17}$$

$$f_i \geq 0, \qquad \forall i \in \{1, n\} \tag{18}$$

Перенумеруем маршруты с учетом преобразований

$$a_1 \leq \cdots \leq a_n \tag{19}$$

Справедлива следующая

Теорема 2. При выполнении условий (12) и (19) в задаче (16) –(18) конкурентное равновесие достигается только когда распределение потока

$$f_i = \begin{cases} \frac{1}{b_i} \cdot t^* - \frac{a_i}{b_i}, & \text{при } i \leq k, \\ 0, & \text{при } i > k, \end{cases} \quad i \in \{1, n\} \tag{20}$$

Где t^* находится, как

$$t^* = \frac{\frac{\mathrm{T}}{r} + \sum_{i=1}^{k} \frac{a_i}{b_i}}{\frac{1}{r} + r \sum_{i=1}^{k} \frac{1}{b_i}} \tag{21}$$

В данной модели, учитывая ограничение (17) , найти количество используемых маршрутов k можно из уравнения

$$\sum_{i=1}^{k} \frac{a_k - a_i}{b_i} < F \tag{22}$$

Доказательство.

Оптимальное время передвижения между Истоком и Стоком является множителем Лагранжа, соответствующим ограничению (17) задачи (16)-(18):

$$L = \sum_{i=1}^{n} \int_0^{f_i} d_i(u)du - \int_0^{F} \frac{1}{u} du + t^* \left(F - \sum_{i=1}^{n} f_i \right) + \sum_{i=1}^{n} \eta_i(-f_i)$$

Где $\eta_i \geq 0$, $i \in \{1, n\}$ – множители Лагранжа, соответствующие ограничению (18) задачи (16)-(18). Дифференцируя данный Лагранжиан по F и f_i, $i \in \{1, n\}$ и приравнивая к нулю, получим

$$d_i(f_i) = t^* + \eta_i, \quad i \in \{1, n\} \tag{23}$$

$$t^* = g^{-1}(F) = T - rF$$

Из условий Куна-Таккера воспользуемся условием дополняющей не жестокости к уравнению (23) и придем к $\eta_i f_i = 0$.

Отсюда,

$$t^* = \begin{cases} a_i + b_i f_i, & \eta_i = 0 \text{ , при } f_i > 0, \\ a_i - \eta_i, & \eta_i > 0 \quad \text{при} f_i = 0, \end{cases} \qquad \forall i \in \{1, n\}$$

Выражаем f_i и получаем

$$f_i = \frac{t^* - a_i}{b_i} \tag{24}$$

Далее подставляя последнее выражение в (17) и выражаем спрос из уравнения (12), находим

$$F = t^* \cdot \sum_{i=1}^{k} \frac{1}{b_i} - \sum_{i=1}^{k} \frac{a_i}{b_i} = \frac{T - t^*}{r}$$

После преобразований получаем (21) и следом (20).

Теорема доказана.

§2.2. Эффективность модели на реальных данных при эластичном спросе

Мы рассматриваем ту же сеть, где Исток- Киркенес, а Сток- Иокогама. Путь из Норвегии в Японию морским ходом может быть пройден либо через Северный морской путь, либо через Суэцкий канал, либо огибая Африку.

По таблице 1, используя входные данные, найдем значения a и b по формулам (14) и (15).

$$a_1 = 0{,}3 \qquad b_1 = \frac{0{,}3}{321} = 0{,}00093$$

$$a_2 = 1{,}13 \qquad b_2 = \frac{1{,}13}{1666} = 0{,}00068$$

$$a_3 = 2 \qquad b_3 = \frac{2}{364} = 0{,}0055$$

В нашей модели возьмем за T такое значение, когда никто из участников движения не захочет передвигаться по этим маршрутам. Зная, сколько наши участники готовы максимально потратить времени на дорогу (T =3), найдем распределение и спрос на этот путь.

Так же надо определить коэффициент r. Для задачи с СМП, этот коэффициент определяется как

$$r = \frac{1}{F}$$

Но так, как он используется в дальнейшем, для нахождения эластичного спроса, то возьмем данные из прошлой главы. Итак, r = 0,0003.

В данной модели мы идем от обратного, находим сначала оптимальное время по формуле (21)

$$t^* = \frac{\frac{3}{0{,}0003}+\frac{0{,}3}{0{,}00093}+\frac{1{,}13}{0{,}00068}+\frac{2}{0{,}0055}}{\frac{1}{0{,}0003}+\frac{1}{0{,}00093}+\frac{1}{0{,}00068}+\frac{1}{0{,}0055}} = \frac{10000+322{,}58+1661{,}76+363{,}64}{3333{,}33+2727{,}68} = 2{,}037$$

По формуле (22) найдем кол-во используемых маршрутов и нижнюю границу спроса, где по прошлой главе мы выяснили, что $k=kue$

$$\frac{2-0{,}3}{0{,}00093}+\frac{2-1{,}13}{0{,}00068}+\frac{2-2}{0{,}0055} = 3107{,}37 < F$$

Далее вычисляем распределение для маршрута *i=1*, через СМП по (20) формуле:

$$f_1 = \frac{1}{0{,}00093} \cdot 2{,}037 - \frac{0{,}3}{0{,}00093} =$$

$$= 2190{,}32 - 322{,}58 = 1867{,}745$$

Найдем распределения для маршрута *i=2*, через Суэцкий канал:

$$f_2 = \frac{1}{0{,}00068} \cdot 2{,}037 - \frac{1{,}13}{0{,}00068} =$$

$$= 2995{,}59 - 1661{,}76 = 1333{,}83$$

Найдем распределения для маршрута i=3, огибая Африку:

$$f_3 = \frac{1}{0{,}0055} \cdot 2{,}037 - \frac{2}{0{,}0055} = 6{,}73$$

По расчетам видно, что условия модели (17), (18) и (22) выполнены. Найдем функции задержек распределенных потоков по формуле (13):

$$d_1(f_1) = 0{,}3 + 0{,}00093 \cdot 1867{,}745 = 2{,}037$$

$$d_2(f_2) = 1{,}13 + 0{,}00068 \cdot 1333{,}83 = 2{,}037$$

$$d_3(f_3) = 2 + 0{,}0055 \cdot 6{,}73 = 2{,}037$$

При заданном T=3, спрос будет 3208. Найденный спрос удовлетворяет условию (22) , значит по всем трём маршрутам пойдут корабли.

Подставив данные значения в уравнение (12), подтверждаем условие (2) и (21):

$$t^* = 3 - 0{,}0003 \cdot 3208{,}305 = 2{,}037$$

$$d_i(f_i) \begin{cases} = 2{,}037, & \text{при } f_i > 0, \\ > 2{,}037, & \text{при} f_i = 0, \end{cases} \qquad \forall i \in \{1{,}3\}$$

Распределение потока из морского порта Норвегии в морской порт Японии за равновесное время (2,037 месяца), при условии, что участники движения готовы потратить 3 месяца на путь:

$$f = (1867{,}745; 1333{,}83; 6{,}73)$$

А спрос будет равен 3208.

В итоге, найдено оптимальное распределение, затраты по всем альтернативным путям одинаковы, по не альтернативным путям равны 0. Если

кто-то поменяет свою стратегию: уйдя с одного маршрута, уменьшит затраты остальных участников, а на тот на который перейдет - увеличит затраты.

Теперь, по методике расчета распределения при эластичном спросе при T=3, рассмотрим другие варианты результатов и показателей участников движения (Таблица 3):

T	***1,5***	***2***	***2,5***	***3***	***5***
к	2	2	2	3	3
t^*	1,212	1,49	1,76	2,037	3,14
F	1102	1809	2497	3209	6214
f	(981; 121;0)	(1280;529; 0)	(1570; 927;0)	(1868;1334;7)	(3051;2956;207)

Таблица 3. Результаты модели с эластичным спросом.

В данной таблице значения *к* показывает, сколько маршрутов в итоге использовано при распределении. В векторах распределений f первое число предполагает распределение по маршруту через Северный Морской путь, второе предполагает количество кораблей по маршруту через Суэцкий канал, а третье число показывает, сколько поплывет кораблей с учетом исходных данных по маршруту, огибая Африку.

Итак, самое главное в модели с эластичным спросом показатель *T.* Исходя из экспертной оценки, которая зависит от психологических ощущений участников движения, можно выяснить: сколько участников готово плыть в таком направлении по таким маршрутам, за какое время при таком спросе и таких ограничениях они поплывут и сколько потратят времени на путь.

Как только Северный Морской путь заработает в полную силу, у всех заинтересованных участников движения изменится этот «порог» *T*, а значит внутреннее ощущение потраченного времени будет влиять на спрос и распределение по маршрутам.

Глава 3. Методические рекомендации по повышению экономической привлекательности СМП.

Пропускная способность Северного морского пути напрямую зависит от количества действующих ледоколов. Данная работа описывает ситуацию, когда СМП работает в полную мощность и все ледоколы ведут за собой как минимум 7 кораблей каравана. По мере того как правильно и качественно заработает маршрут, пороговое значение времени на передвижение по любому направлению будет уменьшаться, а значит будет возрастать спрос на данный маршрут и прибыль страны.

Если на этом пути будет обеспечена необходимая для реального передвижения такого количества судов в месяц пропускная способность, то как минимум 60% потока будет идти через СМП.

Эффективность логистического направления СМП будет реализована только в том случае, когда:

- судоходство будет обеспечено как минимум 9 месяцев в году;
- установлена правильная политика тарифного сбора и условий для получения разрешения грузоперевозок;
- проведена модернизация инфраструктуры (разведка дна, наличие карт, наличие портов на стыке логистических путей и т.д.);
- обеспечена круглосуточная безопасность участников движения;
- возможность предоставления в аренду судов с определенными характеристиками для прохождения пути;
- налажена система предоставления данных о наличии свободного ледокольного сопровождения, а также все показатели для будущей навигации и условий при перевозке (мониторинг и координация судов).

По последнему пункту можно дополнить, что у данного пути нет определенной структуры или института, занимающейся организацией

передвижения, которая могла бы нести ответственность и координировать при возможных случаях отклонения от сроков передвижения. Таким образом, основной рекомендацией служит создание структуры со своим информационным полем (сайт), полномочиями и инвестиционной поддержкой со стороны России и стран входящих в БРИКС.

В дополнение, как рекомендация, предлагается осуществить на будущем информационном поле (сайте), инструмент по которому участник заранее может узнать вероятность отказа ледокольного сопровождения, длину очереди, ориентировочное время передвижения, количество свободных ледоколов. Данный инструмент может осуществляться на основе модели системы массового обслуживания. В приложении будет предоставлен код на языке Java для создания имитационных моделей в среде разработки NetBeans. Настроив показатели под реальные данные, возможно расширить специализацию информационного инструмента и предоставлять данные в зависимости от требований обеих сторон.

Заключение

Целью данной работы была разработка математической модели распределения транспортных потоков с использованием Северного морского пути в качестве национальной единой транспортной коммуникации Российской Федерации в Арктике. Разработано две модели с практическим применением, а именно:

1. Модель распределения транспортного потока с фиксированным спросом.

Результаты этой модели показали, что маршрут огибая Африку у участников движения не должен вызывать интереса, так как велики затраты. Но по реальной ситуации навигации кораблей (Рисунок 4), мы видим обратное. Исходя из этой ситуации, чтобы данное распределение было возможным, нужно внести изменения в саму структуру СМП.

Спрос=3101 кораблей в месяц на путь из Норвегии в Японию. При всех остальных вариациях подбора спроса F, увеличивая его, мы увеличиваем значение функции задержки, а значит и функцию цели.

При таком распределении потока данный путь можно перейти за равновесное время (2 месяца), и это значит, что Северный морской путь будет актуальным маршрутом на сети:

$$f = (1819; 1282;\ 0)$$

2. Модель распределения транспортного потока с эластичным спросом.

Следующая модель отличалась от предыдущей тем, что мы не искали распределение учитывая возможностей сети, а находили спрос и распределения, исходя из предпочтений участников.

Распределение потока из морского порта Норвегии в морской порт Японии за равновесное время (2,037 месяца), при условии, что участники движения готовы потратить 3 месяца на путь:

$$f = (1867{,}745; 1333{,}83; 6{,}73)$$

Найдены также и другие вариации распределения исходя из порога времени *T*. Все они показывают, что модель с эластичным спросом намного адекватнее описывает ситуацию, так как показывает, что при уменьшении психологического порога времени ожидания, Северный Морской путь будет больше принимать кораблей на свою территорию.

Как только Северный Морской путь заработает в полную силу, у всех заинтересованных участников движения изменится этот «порог» *T*, а значит внутреннее ощущение потраченного времени будет влиять на спрос и распределение по маршрутам.

Далее на основе результатов моделей, в 3 главе были предложены рекомендации по повышению экономической привлекательности маршрута СМП для иностранных транзитов.

Проделана большая работа и надо сделать очень много для того, чтобы этот путь был актуальным и безопасным. Данная работа показывает, что будет происходить, как на это повлиять и к чему мы должны стремиться.

Модели описывает ситуацию, когда СМП работает в полную мощность и все ледоколы ведут за собой как минимум 7 кораблей каравана. По мере того как правильно и качественно заработает маршрут, пороговое значение времени на передвижение по любому направлению будет уменьшаться, а значит будет возрастать спрос на данный маршрут и прибыль страны.

Список литературы

1. Комков Н.И., Селин В.С., Цукерман В.А., Горячевская Е.С. Сценарный прогноз развития Северного морского пути // Проблемы прогнозирования. 2016. №2 (155). URL: https://cyberleninka.ru/article/n/stsenarnyy-prognoz-razvitiya-severnogo-morskogo-puti (датаобращения:18.03.2018).
2. Дмитриев Г. Рывок в Арктику// Эксперт Северо-Запада. №39.03.10.11. URL http://expert.ru/northwest/2011/39/ryivok-v-arktiku/ (дата обращения: 18.03.2018).
3. Швецов В. И. Математическое моделирование транспортных потоков // Автоматика и Телемеханика. 2003. № 11.
4. Крылатов А. Ю. Оптимальные стратегии управления транспортными потоками на сети из параллельных каналов // Вестник Санкт-Петербургского университета. Серия 10: Прикладная математика, информатика и процессы управления. 2014. № 2.
5. Захаров В. В., Крылатов А. Ю. Конкурентное равновесие Вардропа на транспортной сети из параллельных неоднородных маршрутов // Процессы управления и устойчивость. 2014. Т. 1. № 1. С. 476–481.
6. Крылатов А. Ю., Шихова К. А. Эластичный спрос на транспортной сети из параллельных маршрутов // Процессы управления и устойчивость. 2016. Т. 3. № 1. С. 736–740.
7. Захаров В. В., Крылатов А. Ю. Системное равновесие транспортных потоков в мегаполисе и стратегии навигаторов: теоретико-игровой подход //Математическая теория игр и ее приложения. 2012. Т. 4. № 4. С. 23–44
8. Wardrop J. G. Some theoretical aspects of road traffic research // Proc.Inst. Civil Eng. 1952. Vol. 1, No 3. P. 325–362

9. Лукин Юрий Федорович Анализ деятельности Северного морского пути // Вестник МГТУ. 2015. №3. URL: https://cyberleninka.ru/article/n/analiz-deyatelnosti-severnogo-morskogo-puti (дата обращения: 24.03.2018).
10. Э.А. Гагарский, начальник Центра транспортной координации и ТТС ОАО «Союзморниипроект», д.т.н., профессор. «Ямал как центр добычи газа». СПБ: Питер, 2014. - 13 с
11. Sheffi Y. Urban transportation networks: equilibrium analysis with mathematical programming methods. N.J.: Prentice-Hall, Inc, Englewood Cliffs, 1985. 416 p.
12. Braess D. Uber ein Paradoxon aus der Verkehrsplanung // Unternehmensforschung 12. 1969. P. 258–268

Приложение

Код для реализации информационного инструмента с комментариями написан на языке Java в среде разработки NetBeans.

Модель 1. Модель системы массового обслуживания с отказами.

```
package queuesimulation;
import java.util.Arrays;
/**
* Модель системы массового обслуживания с отказами.
* Промежутки времени между приходами заявок моделируются
экспоненциальным распределением.
*
* @author Alexandra Marchenko

public class Models1 {

/**
* Возвращает одно значение экспоненциально распределённой случайной
величины
* с заданной средней частотой событий (математическим ожиданием).
* @param lambda Средняя частота событий в единицу времени.
* @return Значение экспоненциально распределённой случайной величины.
*/
public static double randExp(double lambda) {
return -Math.log(1 - Math.random()) / lambda;
}
/**
* Вычисляет вероятность отказа в обслуживании,
* то есть вероятность того, что во время обращения за услугой
* все обслуживающие устройства заняты.
* @param attempts    общее число обращений
* @param numOfUnits   число обслуживающих устройств
* @param avg_interval средний промежуток времени между обращениями
* @param serviceTime  среднее время обслуживания
* @param serviceTimeDifferent разброс времени обслуживания
* @return
*/
private static double taskNunits(int attempts, int numOfUnits,
double avg_interval, double serviceTime, double serviceTimeDifferent) {
//Возвращаемое значение - вероятность отказа в обслуживании.
double ret = 0.;
//Среднее число обращений в единицу времени (частота или интенсивность).
```

```
        double lambda = 1. / avg_interval;
        int busyCount = 0;
//Время, которое осталось до освобождения каждого из приборов.
        double[] units = new double[numOfUnits];
//Изначально все приборы свободны.
        Arrays.fill(units, 0.);
//Начальное обращение занимает начальный прибор на среднее время
обслуживания.
        units[0] = serviceTime;
//Перебор всех обращений
        for (int i = 0; i < attempts; i++) {
//Промежуток времени между обращениями.
        double interval = randExp(lambda);
//Перебор всех приборов и, если прибор занят, то уменьшается время,
//оставшееся время до освобождения прибора на прошедший промежуток
времени.
        for (int j = 0; j < units.length; j++) {
        if (units[j] > 0) {
        units[j] -= interval;
        }
        }
//Блок поиска свободных приборов.
        blockBusy:
        {
//Перебор всех приборов.
        for (int j = 0; j < units.length; j++) {
//Если время, оставшееся до освобождения прибора, неположительное,
//то прибор считается свободным.
        if (units[j] <= 0) {
//Свободному прибору присваивается время занятости,
//равно среднему времени занятости, с добавлением случайной поправки.
//То есть, среднее время занятости равномерно распределено на отрезке
//[serviceTime - serviceTimeDifferent, serviceTime + serviceTimeDifferent]
        units[j] = serviceTime + serviceTimeDifferent * 2. * (0.5 - Math.random());
//Если найден свободный прибор, то другие уже не рассматриваются,
//и производится выход из блока
        break blockBusy;
        }
        }
//Если внутри цикла не было найдено свободного прибора, то
//Счётчик отказов увеличивается на 1.
        busyCount++;
        }
        }
//Отношение числа отказов к числу обращений - вероятность отказов.
```

```
        ret = (double)busyCount / (double)attempts;
        return ret;
        }
/**
* Запускает модель системы массового обслуживания
*/
        private static void test_TaskNUnits(){
        int n = 100;
        double dt = 1;
        int    nu = 7;
        double srv = 8;
        double srv_diff = 1;
        System.out.println("------------------------------"
+ "\r\n Число обращений (заявок): " + n
+ "\r\n количество приборов или ледоколов: " + nu
+ "\r\n средний промежуток между обращениями 1 день: " + dt
+ "\r\n среднее время обслуживания 8 дней: " + srv
+ "\r\n разброс времени обслуживания: " + srv_diff
+ "\r\n вероятность отказа в обслуживании: " + taskNunits(n, nu, dt, srv, srv_diff));
            }
        public static void main(String[] args) {
        test_TaskNUnits();
        }
        }
```

Модель 2. Модель системы массового обслуживания с ожиданием.

```
        package queuesimulation;
        import java.util.Arrays;
/**
* @author Alexandra Marchenko
*/
        public class Models2
        private static QueueStatistics taskNunits(
        int attempts, int numOfUnits, Incoming inc, Waiting queue) {
//Возвращаемое значение - сводные данные по системе массового обслуживания.
        QueueStatistics ret =  new QueueStatistics();
//Текущее время
        double curTime = 0.;
//Суммарное время ожидания обслуживания.
        double sumWaitingTime = 0.;
//Суммарное время простоя приборов
        double[] sumUnitsFreeTime = new double[numOfUnits];
        Arrays.fill(sumUnitsFreeTime, 0.);
```

```
//Счётчик числа отказов в обслуживании.
        int busyCount = 0;
//Время, которое осталось до освобождения каждого из приборов.
        double[] units = new double[numOfUnits];
//Изначально все приборы свободны.
        Arrays.fill(units, 0.);
//Начальное обращение (заявка на обслуживание) из входящего потока
        CallReq req = inc.nextCall();
        //Начальное обращение занимает начальный прибор на своё время
обслуживания.
        units[0] = req.amount;

//Перебор всех обращений
        for (int i = 1; i < attempts; i++) {
//Очередное обращение
        req = inc.nextCall();
//Промежуток времени до следующего обращения.
        double interval = req.time; curTime += interval;
//Перебор всех приборов. Уменьшается время,
//оставшееся время до освобождения прибора на прошедший промежуток
времени.
        for (int j = 0; j < units.length; j++) {
        units[j] -= interval;
        }
//Блок поиска свободных приборов.
        blockBusy:
        {
//Переупорядочивание приборов в порядке освобождения
        int[] unitsNums = sortFreeUnits(units);
//Перебор всех приборов в порядке освобождения.
        for (int j = 0; j < unitsNums.length; j++) {
//Если время, оставшееся до освобождения прибора, неположительное,
//то прибор считается свободным.
        if (units[unitsNums[j]] <= 0) {
        if(queue.size() > 0) {
//В очереди есть заявки - заявка поступает в обработку
        CallReq servReq = queue.get(0);
        units[unitsNums[j]] += servReq.amount;
//Заявка удаляется из очереди
        queue.remove(0, curTime);
        sumWaitingTime += servReq.getWaitingTime();
        }
        else {
//Очередь пуста.
//Пришедшая заявка поступает в обработку.
```

```
//Учитывается время простоя прибора.
   sumUnitsFreeTime[unitsNums[j]] -= units[unitsNums[j]];
//Прибору присваивается время занятости.
        units[unitsNums[j]] = req.amount;
//Поскольку найден свободный прибор для пришедшей заявки,
//то производится выход из блока
        break blockBusy;
        }
        }
        }
//Если внутри цикла не было найдено свободного прибора для поступившей
заявки,
//то выхода из блока не было.
//Делается попытка поставить заявку в очередь
        if(!queue.add(req, curTime)){
//Если очередь заполнена, то счётчик отказов увеличивается на 1.
        busyCount++;
        }
        }//blockBusy
//Необходимо обнулить время свободных приборов, если такие есть.
        for (int j = 0; j < units.length; j++) {
        if(units[j] < 0) {
        sumUnitsFreeTime[j] -= units[j];
        units[j] = 0;
        }
        }
        }
        double sumUnitsFreePartTime = 0;
        for (int i = 0; i < sumUnitsFreeTime.length; i++) {
        sumUnitsFreePartTime += (sumUnitsFreeTime[i] /= curTime);
        }
//Отношение числа отказов к числу обращений - вероятность отказов.
        ret.rejectProbability = (double)busyCount / (double)attempts;
        ret.avarageWaitingTime = sumWaitingTime /  (double)attempts;
        ret.unitsFreePartTime = sumUnitsFreeTime;
        ret.unitsSumFreePartTime = sumUnitsFreePartTime;
        return ret;
        }
/**
* Переупорядочивает приборы в порядке их освобождения
* @param units
* @return
*/
        public static int[] sortFreeUnits(double[] units) {
        int[] freeUnitsNums = new int[units.length];
```

```
        for (int i = 0; i < freeUnitsNums.length; i++) {
        freeUnitsNums[i] = i;
        }
//Повторяется сортировка методом пузырька
        for (int j = 0; j < freeUnitsNums.length - 1; j++) {
//Самый поздний прибор из оставшихся окажется в конце
        for (int i = 0; i < freeUnitsNums.length - j - 1; i++) {if (units[freeUnitsNums[i]] >
units[freeUnitsNums[i + 1]]) {int ti = freeUnitsNums[i];
        freeUnitsNums[i] = freeUnitsNums[i + 1];
        freeUnitsNums[i + 1] = ti;
        }
        }
        }
        return freeUnitsNums;
        }
/**
* Запускает модель системы массового обслуживания
*/
        private static void test_TaskNUnits(){
        int n = 100;
        int nu = 7;
        double lambda = 3;
        double srv = 8;
        int waitingRoomSize = 10000000;
        Incoming income = new Incoming();
        income.timeIntervals = ExponentialSequence.createByMean(1. / lambda);
        income.reqAmounts = UniformSequence.createByMeanAndDeviation(srv, srv *
0.3);
        income.agreeToWaitProbability = 1;
        Waiting queue = new Waiting();
        queue.capacity = waitingRoomSize;
        QueueStatistics res = taskNunits(n, nu, income, queue);
        System.out.println("--------------------------------"
+ "\r\n Число обращений: " + n
+ "\r\n количество приборов: " + nu
+ "\r\n среднее число обращений в единицу времени: " + lambda   //интенсивность
потока
+ "\r\n среднее время обслуживания: " + srv
+ "\r\n число мест для ожидания: " + waitingRoomSize
+ "\r\n среднее время ожидания: " + res.avarageWaitingTime
+ "\r\n вероятность отказа в обслуживании: " + res.rejectProbability
+ "\r\n относительный простой всех приборов: " + res.unitsSumFreePartTime
+ "\r\n средний относительный простой одного прибора: " +
(res.unitsSumFreePartTime / nu)
        );
```

```
        for (int i = 0; i < res.unitsFreePartTime.length; i++) {System.out.println( "[" + i +
"]  " + res.unitsFreePartTime[i]);
        }
        }
        public static void main(String[] args) {
        test_TaskNUnits();
        }
        public static class QueueStatistics {
        public double rejectProbability;
        public double avarageWaitingTime;
        public double[] unitsFreePartTime;
        public double unitsSumFreePartTime;
        }
        }
```

Дополнения к модели 2.

```
        package queuesimulation;
/**
* Модель входящего потока.
        public class Incoming {
/**
* Последовательность временных промежутков между заявками.
*/
        public RandomSequence timeIntervals;
/**
* Размер заявки, например, время, необходимое для её обработки.
*/
        public RandomSequence reqAmounts;
 /**
* Вероятность того, что заявка будет ждать обслуживания при наличии
очереди.
*/
        public double agreeToWaitProbability = 1;

/**
* Возвращает очередную заявку
* @return заявка
*/
        public CallReq nextCall(){
        CallReq req = new CallReq();
        req.time = timeIntervals.getNext();
        req.amount = reqAmounts.getNext();
        req.type = Math.random() > agreeToWaitProbability ?0:1;
        return req;
```

```
};
}
```

……

```
package queuesimulation;
import java.util.ArrayList;

/**
 * Модель очереди заявок, ожидающих обслуживания.

 */
public class Waiting {

 public static final int UNLIMITED = -1;

 /**
 * Maximal length of reqs queue
 */
   public int capacity = 0;

 /**
 * Maximal waiting time in queue
    */
   public int duration;

 /**
 * Queue
    */
   private ArrayList<CallReq> reqs = new ArrayList<CallReq>();

 private double lastLengthChangeTime = 0;
 private double filling = 0;
 private double avarageLength = 0;
 private double maxLength = 0;
 /**
 * Добавляет заявку в очередь.
 * @param req call-req to be added to the end of queue
 * @param curTime current time
 * @return
    */
   public boolean add(CallReq req, double curTime){
 if((capacity == UNLIMITED || reqs.size() < capacity) && (req.type > 0 /*|| reqs.size()
== 0*/)) {
 if(!reqs.add(req)) {
```

```
throw new RuntimeException("CallReq instance can't be added to queue");
}
if(reqs.size() > maxLength) maxLength = reqs.size();
req.startWaitTime = curTime;
filling += (reqs.size() - 1) * (curTime - lastLengthChangeTime);
avarageLength = filling / curTime;
lastLengthChangeTime = curTime;
return true;
} else {
return false;
}
}
/**
* Удаляет заявку из очереди.
* @param i number of call-req in queue
* @param curTime current time
   */
  public CallReq remove(int i, double curTime) {
CallReq req =reqs.get(i);
req.finishWaitTime = curTime;
reqs.remove(i);
filling += (reqs.size() + 1) * (curTime - lastLengthChangeTime);
avarageLength = filling / curTime;
lastLengthChangeTime = curTime;
return req;
}

/**
* Возвращает длину очереди.
* @return queue length
   */
  public int size(){
return reqs.size();
}
/**
* Возвращает заявку по номеру очереди.
* @param i number of call-req in queue
* @return call-req number i
   */
  public CallReq get(int i) {
return reqs.get(i);
}
/**
* Очищает очередь от всех заявок.
   */
```

```
   public void clear(){
 lastLengthChangeTime = 0;
 filling = 0;
 avarageLength = 0;
 maxLength = 0;
 reqs.clear();
 }
 /**
 * Возвращает среднюю длину очереди.
 * @return avarage queue length
    */
   public double getAvarageLength() {
 return avarageLength;
 }
 /**
 * Возвращает наибольшую длину очереди.
 * @return maximal queue length
    */
   public double getMaxLength() {
 return maxLength;
 }
}
```

```
package queuesimulation;

/**
 * Обращение-заявка на обслуживание.
 * С усложнением модели возможно добавятся дополнительные свойства класса.
 */
public class CallReq {

   public double time;
   public int type;
   public double amount;
   public double startWaitTime;
   public double finishWaitTime;

   public double getWaitingTime(){
      return finishWaitTime - startWaitTime;
   }
}
```

```
package queuesimulation;

/**
 * Абстрактный класс для описания различных случайных величин в классах-потомках.
 *
 */
public abstract class RandomSequence {

    public abstract double getNext() ;

}
```

……

```
package queuesimulation;

/**
 * Задаёт равномерно распределённую случайную последовательность чисел на отрезке [0,1).

 */
public class UniformSequence extends RandomBaseSequence{

    /**
     * Множитель случайной величины.
     */
    private double multiplier = 1.;
    /**
     * Смещение случайной величины.
     */
    private double increment = 0.;


    private UniformSequence(double mean) {
        super(mean, mean * mean / 12.);
        multiplier = mean * 2.;
    }

    private UniformSequence(double mean, double deviation) {
        super(mean, deviation * deviation / 3.);
        multiplier = deviation * 2.;
        increment = mean - deviation;
    }
```

```
    private UniformSequence(double mean, long seed) {
        super(mean, mean * mean / 12., seed);
        multiplier = mean * 2.;
    }

    private UniformSequence(double mean, double deviation, long seed) {
        super(mean, deviation * deviation / 3., seed);
        multiplier = deviation * 2.;
        increment = mean - deviation;
    }

    /**
     * Создание по мат.ожиданию (разброс от нуля до двух мат.ожиданий)
     * @param mean
     * @return
     */
    public static UniformSequence createByMean(double mean){
        return new UniformSequence(mean);
    }

    /**
     * Создание по мат.ожиданию и наибольшему отклонению от мат.ожидания
     * @param mean
     * @param deviation
     * @return
     */
    public static UniformSequence createByMeanAndDeviation(double mean, double deviation){
        return new UniformSequence(mean, deviation);
    }

    /**
     * Создание по мат.ожиданию и дисперсии
     * @param mean
     * @param variance
     * @return
     */
    public static UniformSequence createByMeanAndVariance(double mean, double variance){
        return new UniformSequence(mean, Math.sqrt(3 * variance));
    }

    /**
```

```
     * Создание по границам изменения
     * @param min
     * @param max
     * @return
     */
    public static UniformSequence createByInterval(double min, double max){
        return new UniformSequence((max + min) / 2., (max - min) / 2.);
    }


    /**
     * Создание по мат.ожиданию (разброс от нуля до двух мат.ожиданий)
     * с числом, порождающим случайную последовательность.
     * @param mean
     * @return
     */
    public static UniformSequence createByMeanWithSeed(double mean, long seed){
        return new UniformSequence(mean, seed);
    }


    /**
     * Возвращает очередное значение случайной величины.
     * @return очередное значение случайной величины.
     */
    @Override
    public double getNext() {
        return getRand().nextDouble() * multiplier + increment;
    }

}
```

```
package queuesimulation;

/**
 * Последовательность значений экспоненциально распределённой
случайной величины.
*/
public class ExponentialSequence  extends RandomBaseSequence{

 private ExponentialSequence(double mean) {
 super(mean, mean * mean);
 }
```

```
private ExponentialSequence(double mean, long seed) {
super(mean, mean * mean, seed);
}

/**
* Создаёт сущность класса по математическому ожиданию.
* @param mean математическое ожидание
* @return
   */
   public static ExponentialSequence createByMean(double mean){
return new ExponentialSequence(mean);
}

/**
* Создаёт сущность класса по математическому ожиданию и числу, порождающему случайную последовательность.
*
* @param mean математическое ожидание
* @param seed число, порождающее случайную последовательность
* @return
   */
   public static ExponentialSequence createByMeanWithSeed(double mean, long seed){
return new ExponentialSequence(mean, seed);
}

/**
* Возвращает очередное значение случайной величины.
* @return очередное значение случайной величины.
   */
@Override
public double getNext() {
return -Math.log(getRand().nextDouble()) * getMean(); //-Math.log(1 - getRand().nextDouble()) * getMean();
}

/**
* Проверка работы класса.
* @param args Не используется.
   */
   public static void main(String[] args) {
~~oldTest~~();
newTest();
}
```

```
 private static void newTest() {
 int attempts = 10000000;
 Histogram histo = new Histogram(10, 0, 0.1);
 ExponentialSequence expSeq = new ExponentialSequence(.5);
 double sum = 0;
 double sum2 = 0;
 for (int i = 0; i < attempts; i++) {
 double value = expSeq.getNext();
 sum += value;
 sum2 += value * value;
 histo.put(value);
 }
 System.out.println(histo);
 double avg = sum / attempts;
 double var = avg * avg * attempts - 2 * avg * sum + sum2;
 var /= attempts;
 System.out.println("avg: " + avg + " var: " + var);
 }

private static void oldTest() {
 double[] data = new double[10000000];
 ExponentialSequence expSeq = new ExponentialSequence(.5);
 double sumAvg = 0;
 for (int i = 0; i < data.length; i++) {
 data[i] = expSeq.getNext();
 sumAvg += data[i];
 }
 sumAvg /= data.length;
 int[] histo = ru.spbu.apmath.class469.First.makeHistogram(10, 0, 0.1, data);
 ru.spbu.apmath.class469.First.printHisto(0, 0.1, histo);
 double var = 0;
 for (int i = 0; i < data.length; i++) {
 var += (sumAvg - data[i]) * (sumAvg - data[i]);
 }
 var /= data.length;
 System.out.println("avg: " + sumAvg + " var: " + var);
 }
 }
```

```
package queuesimulation;
import java.util.ArrayList;

/**
```

*** Модель очереди заявок, ожидающих обслуживания.**

```
 */
public class Waiting {

 public static final int UNLIMITED = -1;

 /**
 * Maximal length of reqs queue
    */
   public int capacity = 0;

 /**
 * Maximal waiting time in queue
    */
   public int duration;

 /**
 * Queue
    */
   private ArrayList<CallReq> reqs = new ArrayList<CallReq>();

 private double lastLengthChangeTime = 0;
 private double filling = 0;
 private double avarageLength = 0;
 private double maxLength = 0;
 /**
 * Добавляет заявку в очередь.
 * @param req call-req to be added to the end of queue
 * @param curTime current time
 * @return
    */
   public boolean add(CallReq req, double curTime){
 if((capacity == UNLIMITED || reqs.size() < capacity) && (req.type > 0 /*|| reqs.size()
== 0*/)) {
 if(!reqs.add(req)) {
 throw new RuntimeException("CallReq instance can't be added to queue");
 }
 if(reqs.size() > maxLength) maxLength = reqs.size();
 req.startWaitTime = curTime;
 filling += (reqs.size() - 1) * (curTime - lastLengthChangeTime);
 avarageLength = filling / curTime;
 lastLengthChangeTime = curTime;
 return true;
 } else {
 return false;
 }
```

```
}

 /**
* Удаляет заявку из очереди.
* @param i number of call-req in queue
* @param curTime current time
   */
  public CallReq remove(int i, double curTime) {
CallReq req =reqs.get(i);
req.finishWaitTime = curTime;
reqs.remove(i);
filling += (reqs.size() + 1) * (curTime - lastLengthChangeTime);
avarageLength = filling / curTime;
lastLengthChangeTime = curTime;
return req;
}

/**
* Возвращает длину очереди.
* @return queue length
   */
  public int size(){
return reqs.size();
}

/**
* Возвращает заявку по номеру очереди.
* @param i number of call-req in queue
* @return call-req number i
   */
  public CallReq get(int i) {
return reqs.get(i);
}

/**
* Очищает очередь от всех заявок.
   */
  public void clear(){
lastLengthChangeTime = 0;
filling = 0;
avarageLength = 0;
maxLength = 0;
reqs.clear();
}
```

```
/**
* Возвращает среднюю длину очереди.
* @return avarage queue length
   */
  public double getAvarageLength() {
return avarageLength;

/**
* Возвращает наибольшую длину очереди.
* @return maximal queue length
   */
  public double getMaxLength() {
return maxLength;
}
}
```

```
package queuesimulation;
import java.util.Random;
/**
 * Абстрактный класс для описания различных случайных величин в классах-потомках.
 * Содержит равномерно распределённую случайную величину для порождения других случайных величин.
 *
 */
public abstract class RandomBaseSequence extends RandomSequence {
   private final Random rand;
   private double mean = 0;
   private double variance = 1;

   public RandomBaseSequence() {
      rand = new Random();
      mean = 0;
   }

   public RandomBaseSequence(double mean, double variance) {
      this.mean = mean;
      this.variance = variance;
      //rand = new Random(new java.util.Date().getTime());
      rand = new Random();
   }

   public RandomBaseSequence(long seed) {
      rand = new Random(seed);
```

```
    }

    public RandomBaseSequence(double mean, double variance, long seed) {
        rand = new Random(seed);
        this.mean = mean;
        this.variance = variance;
    }
    protected Random getRand() {
        return rand;
    }

    public double getMean() {
        return mean;
    }
    public void setMean(double mean) {
        this.mean = mean;
    }
    public double getVariance() {
        return variance;
    }
    public void setVariance(double variance) {
        this.variance = variance;
    }

}
```

Как в итоге должен выглядеть результат модели: