Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

“КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ”

Институт математики и механики им. Н.И. Лобачевского

Кафедра теории функции и приближений

Направление: 01.03.01 — математика

Выпускная квалификационная работа

(бакалаврская работа)

Применение муравьиных алгоритмов для моделирования оптимального туристического маршрута в ГИС

Работа завершена:

«___» __________ 2015 г. ________________(Макаров Р. В.)

Работа допущена к защите:

Научный руководитель

к.п.н., ст. преподаватель

«___» __________ 2015 г. _______________(Маклецов С. В.)

Заведующий кафедрой

д. физ.-мат. наук, профессор

«___» __________ 2015 г. _______________(Авхадиев Ф.Г.)

КАЗАНЬ — 2015

Содержание

Введение 3
1. Геоинформационные системы 6
2. Математическая модель решения задачи поиска оптимального маршрута 8
Муравьиные алгоритмы 11
Особенности решения задачи поиска оптимального маршрута 16
3. Программная реализация модифицированного муравьиного алгоритма ... 20
Заключение 24
Список литературы. 25
Приложения 29

Введение

Актуальность темы. В последнее время перед различными регионами России, и, в частности, перед Татарстаном, возникают перспективы увеличения потока туристов, как внутрироссийских, так и иностранных. Во-первых, это происходит за счет проведения крупных международных мероприятий, таких как, Универсиада (2013 г.), чемпионат мира по водным видам спорта (2015 г.), чемпионат мира по футболу (2018 г.) и др. Во-вторых, этому будет способствовать реализация ряда обсуждаемых проектов по улучшению инфраструктуры, таких, например, как строительство высокоскоростной железнодорожной магистрали Москва-Пекин. Все это, безусловно, требует развития гостиничного комплекса и сферы въездного туризма в целом. Однако увеличившаяся значимость информационных технологий в жизни современного общества приводит к необходимости разработки и воплощения в жизнь электронных систем, направленных на повышение привлекательности Татарстана для посещения гостями из других стран и регионов России.

Одним из способов повышения туристической привлекательности региона через сеть Интернет является создание программного обеспечения, позволяющего организовать экскурсионный маршрут в городе через основные достопримечательности с учетом имеющегося у туриста времени и предпочтений. Это требует применения определенных алгоритмов для поиска соответствующего пути следования. В настоящей работе рассматривается возможность применения так называемых природных вычислений и, в частности, муравьиных алгоритмов для решения поставленной задачи. Данной проблемой серьезно начали заниматься европейские ученые с середины 90-х годов. Первая версия алгоритма была предложена доктором наук Марко Дориго в 1992 году в работах [1] и [2] и была направлена на поиск оптимального пути в графе. Дальнейшими

исследованиями в этой области занимались Бьянки Л. [3], Лореш М.А.[4], Дудин П.А.[5], Глушко С.И.[6] и др.

Рассматриваемая в работе тема имеет практическую и теоретическую значимость. Практическая важность заключается в

Цель работы — разработать алгоритм поиска оптимального туристического маршрута с применением математических методов, основанных на природных механизмах поиска наилучшего решения, его теоретическое и экспериментальное исследование.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1) изучить возможности геоинформационных систем (ГИС) для хранения, обработки необходимого массива данных, а также для предоставления удобного пользовательского интерфейса;
2) создать математическую модель решения задачи построения оптимального маршрута, позволяющего посетить как можно больше вершин графа с учетом их приоритета и ограничений по времени;
3) разработать программное обеспечение, способное по задаваемым туристом требованиям осуществить расчет и наглядно представить искомый маршрут.

Теоретическая значимость работы заключается том, что в ней предложена математическая модель решения задачи поиска оптимального маршрута в графе при заданных временных ограничениях и приоритетах переходов, а также в модификации муравьиного алгоритма отыскания соответствующего пути.

Практическая значимость исследования состоит в разработке приложения, содержащего графический интерфейс пользователя который

посредством программной связи с ГИС на основе построенной модели позволяет решить поставленную задачу.

Объектом исследования являются природные механизмы решения задач оптимизации.

Предметом исследования является вариант муравьиного алгоритма поиска оптимального туристического маршрута в ГИС.

1. Геоинформационные системы

Во все времена знания о пространственной ориентации физических объектов или, попросту говоря, об их географическом положении, были очень важны для людей. К примеру, первобытные охотники всегда знали местонахождение своей добычи, а жизнь или смерть исследователей-первопроходцев напрямую зависела от их знаний географии. Также и современное общество живет, работает и сотрудничает, опираясь на информацию о том, кто и где находится. Прикладная география в виде карт и информации о пространстве помогала совершать открытия, способствовала торговле, повышала безопасность жизнедеятельности человечества в течение как минимум последних 3000 лет, а карты являются одними из наиболее красивейших документов, рассказывающих об истории нашей цивилизации [7, с. 9].

Необходимость проанализировать картографические данные возникает у людей различных профессий, владеющих большими объемами информации, на основе которых принимаются решения. В таких случаях, как правило, применяются геоинформационные системы [8, с. 5].

В статье [9] приводится следующее определение: «ГИС — это современная компьютерная технология для картирования и анализа объектов реального мира, а также событий, происходящих на нашей планете».

Появление ГИС относят к началу 60-х гг. XX в., когда появились предпосылки и условия для информатизации и компьютеризации сфер деятельности, связанных с моделированием пространства и решением пространственных задач [10, с. 11] .

ГИС интегрирует в себе системы сбора информации (зондирование, мониторинг), ее хранения (базы данных, информационно-поисковые

системы), обработки (моделирование, обработка изображений) и отображения (графика, электронные карты) [11, с.16].

Основная идея ГИС — связать информацию на карте с базами данных. При этом такие системы включают аналитические средства для работы с любой координатно привязанной информацией. ГИС предлагает новый путь развития картографии, преодолевая недостатки бумажных карт — статичность и ограниченную емкость, потому как в последние десятилетия карты из-за большого объема данных перестают быть простыми для восприятия. В то же время, в электронном представлении вся информация хорошо визуализирована: есть возможность получать сведения по заданным параметрам, не сильно нагружая карту ненужной информацией, тем самым ускоряя производительность и улучшая эффективность обработки и анализа [8, с. 15].

В нашей работе мы будем использовать ГИС, разработанную кампанией Яндекс — «Яндекс.Карты», поисково-информационный картографический сервис, открытый в 2004 году. На сервисе представлены подробные карты всего мира и, в частности, г. Казани.

Данную ГИС мы будем использовать как для задания всех необходимых условий, так и для вывода на карте построенного нашей программой оптимальной траектории движения туриста для посещения достопримечательностей.

Расчет маршрута, начинающегося и заканчивающегося в указанных точках и проходящего через знаменательные места, включая те, которые пользователь отметил как приоритетные, с учетом отведенного времени, требует формирования графа.

2. Математическая модель решения задачи поиска оптимального маршрута

Теория графов является эффективным аппаратом формализации современных задач и, несмотря на кажущуюся простоту, является одним из элегантных разделов математики с широкой областью применения. В частности, она может быть использована при проведении системных исследований, проектировании систем логического управления и баз данных, а также решении задач поиска оптимального маршрута.

Графом называется «пара множеств (V,E), где E — произвольное подмножество из $V^{(2)}$. Элементы множеств V и E называют соответственно вершинами и ребрами графа» [12, с. 6].

Модель поиска оптимального туристического маршрута представима в виде графа, вершинами которого будут являться сами достопримечательности, а также начальная и конечная точки. В качестве весов на ребрах мы возьмем время, необходимое для перехода от одной точки до другой.

Граф, лежащий в основе математической модели, должен быть полным, то есть все его вершины должны быть соединены между собой. Этого требует постановка исходной задачи, поскольку у туриста должна быть возможность добраться от одной достопримечательности до любой другой.

В задаче вершины, включая начальную и конечную точки, различны, а потому граф является открытым и маршрут является цепью [13, с.17].

Проблема поиска оптимального маршрута имеет много общего с классической задачей теории графов — задачей коммивояжера. Она известна человечеству с давних времен, но до сих пор является весьма актуальной, в том числе и для современной алгоритмистики [14, с. 3].

Сама задача формулируется довольно просто: имеется n городов, расстояния между которыми нам известны. Коммивояжер находится в одном из этих городов, и желает объехать все города, при этом посетив каждый из них ровно один раз, а затем вернуться в исходный город. Кроме того, искомый путь должен иметь наименьшую длину [15, с. 33-34].

Простота постановки задачи о коммивояжере сочетается с чрезвычайной трудностью ее решения, причем трудности вычислительного характера, так как способ подсчета, сам по себе, довольно прост: перебрать все возможные варианты маршрутов и выбрать кратчайший. Но все дело в том, что при сколько-нибудь значительном числе городов, ни одна, даже самая быстродействующая вычислительная машина не в состоянии осуществить такой перебор за приемлемое время [16, с. 4-5].

Теорема. Задача коммивояжера является NP-трудной, даже когда матрица расстояний между городами симметрична [17, с. 43].

Действительно, находясь в одном из городов, коммивояжер может направиться в один из (n-1) оставшихся городов, откуда далее пойти в один из (n-2) городов и т. д., до тех пор, пока не останется один город, посетив который он направится в город, из которого он вышел. Тогда количество способов равно (n-1)(n-2)…2·1=(n-1)!, что является довольно большим числом (по формуле Стирлинга $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, а значит скорость функции при увеличении n очень высокая).

В таком случае следует отказаться от попыток отыскать точное решение задачи коммивояжера и сосредоточиться на поиске приближенного — пускай не оптимального, но хотя бы близкого к нему. В виду большой практической важности проблемы, полезными будут и приближенные решения [18].

Существует множество различных способов решения этой задачи, как точных так и приближенных. В качестве примера точного метода ее решения можно привести такие, как честный перебор — перебор всевозможных вариантов и в дальнейшем выбор среди них оптимального.

Все методы, сокращающие полный перебор, являются приближенные, соответственно в них ищется не точный маршрут, а лишь приближенный к нему. Наиболее популярными из таких методов являются: жадный алгоритм, модификация алгоритма Дейкстры, метод ветвей и границ, генетический алгоритм, имитационный отжиг, муравьиные алгоритмы.

Так, например, в соответствии с жадным алгоритмом поиск оптимального пути начинается со случайной вершины, после чего на каждом шаге осуществляется переход в город, который находится ближе всех к текущему. При этом поиск выходит сравнительно быстрым, однако погрешность может оказаться довольно большой [17, с. 46].

В основе модифицированного алгоритма Дейкстры лежит классический: ищется кратчайший маршрут от одной вершины до любой другой посредством отбрасывания по ходу решения ненужных нам путей. Отличие измененного варианта алгоритма заключается в том, что в нем осуществляется поиск пути, проходящего через как можно большее число вершин, с последующим возвратом в исходную [19, с. 235].

Метод ветвей и границ использует улучшенный перебор, откидывая на каждом шаге алгоритма явно неоптимальные решения (для небольшого числа вершин точность выходит хорошая, однако с увеличением городов, ответ будет далек от правды) [15, с. 96-97].

Генетический алгоритм, используемый для решения задач оптимизации и моделирования случайным подбором, заключается в поиске путем комбинирования и вариаций параметров, напоминающих биологическую эволюцию. Отличительной способностью алгоритма является использование

операторов скрещивания и мутации, производящих рекомбинацию решений-кандидатов, аналогичной роли скрещивания в живой природе [20, с. 356-357].

Имитационный отжиг является одним из методов решения задачи оптимизации, в основе которого лежит процесс остывания твердого вещества, при котором молекулы на фоне снижающейся со временем скорости движения собираются в наиболее выгодные, в плане энергии, конструкции [18].

В таблице 1 [21] приведены найденные различными методами длины маршрутов для задачи коммивояжера с различным числом городов (жирным шрифтом выделены наилучшие полученные результаты).

Таблица 1.

Сравнение эврестических методов оптимизации маршрута коммивояжера

	Задача коммивояжера для 50 городов	Задача коммивояжера для 75 городов	Задача коммивояжера для 100 городов
Имитационный отжиг	443	580	нет данных
Генетические алгоритмы	428	545	22761
Муравьиные алгоритмы	**425**	**535**	**21282**

Нетрудно заметить, что муравьиные алгоритмы позволяют получить приближенные решения с наименьшей погрешностью. Потому для решения задачи мы будем использовать именно этот алгоритм.

Муравьиные алгоритмы

В последние годы интенсивно разрабатывается научное направление Natural Computing («Природные вычисления»), объединяющее математические методы, в которых заложены принципы природных механизмов принятия решений. Эти механизмы обеспечивают эффективную

адаптацию флоры и фауны к окружающей среде на протяжении миллионов лет.

Одним из таких методов являются муравьиные алгоритмы, в основе которых лежат принципы самоорганизации муравьиной колонии. Несмотря на разобщенное поведение каждого из своих представителей, она образует высокоорганизованную систему, состоящую из большого количества «агентов» — муравьев, и благодаря этому способна решать сложные задачи, которые превышают способности каждого отдельного своего элемента. Изучение поведения колонии муравьев интересно для компьютерных наук, поскольку дает представление о разобщенной организации, что очень полезно для решения сложных задач оптимизации [22, с. 851].

Поведение муравьев в природе

Поведение колонии муравьев обеспечивает достижение общих целей на основе низкоуровнего взаимодействия. Колония не имеет централизованного управления, и обмен информацией между ее особями происходит локально, то есть между отдельными ее представителями, или же непрямым обменом, который и лежит в основе муравьиных алгоритмов [23, с. 1].

Непрямой обмен представляет собой взаимодействие между муравьями путем изменения некоторой области окружающей среды с помощью специального химического вещества — феромона — секрета специальных желез, откладываемых муравьями при перемещении в пространстве. Концентрация феромона на пути определяет предпочтительность движения по данному направлению [24, с. 4].

Адаптивность поведения муравьев реализуется испарением феромона в пространстве в течение некоторого времени. Поэтому мы можем провести некоторую аналогию между распределением феромона в окружающем

колонию пространстве, и «глобальной» памятью муравейника, носящий динамический характер [23, с. 11-12].

Идея муравьиного алгоритма

Идея муравьиного алгоритма заключается в моделировании поведения муравьев, связанного с их способностью находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь. При своем движении муравей метит свой путь феромоном, и эта информация используется другими муравьями. Это и является главным правилом поведения каждого из представителей колонии в случае, если старый маршрут стал недоступен. Если во время движения муравей столкнулся с преградой, то с равной вероятностью он обойдет ее слева или справа. То же самое произойдет и на обратном пути. Однако, те муравьи, которые выберут кратчайший путь, будут проходить его быстрее и за несколько перемещений он будет сильнее обогащен феромоном. Поскольку его количество влияет на выбор пути муравьями, то вскоре подавляющее большинство представителей колонии будут предпочитать этот маршрут. Однако испарение пахучего следа в воздухе гарантирует, что найденное локально-оптимальное решение не будет единственным, и муравьи будут продолжать искать и другие пути. Если мы моделируем процесс такого поведения на графе, ребра которого — всевозможные пути движения, то в течение некоторого времени наиболее обогащенный феромоном путь и будет наикратчайшим, что и даст решение задачи [20, с. 362].

Способ применения муравьиного алгоритма

Рассмотрим пример решения задачи коммивояжера. Классический вариант, применяемого с этой целью муравьиного алгоритма, описан в [21] и выглядит следующим образом.

Многократность выполнения действия осуществляется при помощи итераций одновременно несколькими муравьями. За одну итерацию каждый муравей проходит маршрут через все города, независимо от других.

Для муравья, находящегося в городе i, переход в город j зависит от следующих трех составляющих: памяти муравья, видимости и виртуального следа феромона.

Память муравья — это список тех городов, в которых данный муравей уже побывал и заходить в которые ему еще раз нельзя. Наличие этого списка гарантирует, что муравей не посетит дважды какой-либо из городов. Ясно, что с движением муравья этот список возрастает и будет обнуляться в начале каждой итерации. За $J_{i,k}$ принимается список городов, которые муравей k, находящийся в городе i еще не посетил. Понятно, что $J_{i,k}$ является дополнением к памяти муравья.

Видимость — величина, обратная расстоянию: $\eta_{ij}=1/D_{ij}$, где D_{ij} — расстояние между городами i и j. Видимость — это информация, выражающая желание муравья посетить город j из города i. Чем ближе город, тем больше желание посетить его. Однако для нахождения кратчайшего маршрута использование одной лишь видимости недостаточно.

Виртуальный след феромона на пути между городами i и j являет собой подтвержденное опытом муравьев желание посетить город j из города i. И в отличие от видимости эта величина изменяется после каждой из итераций алгоритма. За количество виртуального феромона на пути из города i в город j на итерации t берется $\tau_{ij}(t)$.

Кроме того, важную роль в алгоритме играет вероятностно-пропорциональное правило, которое определяет вероятность перехода k-го муравья из города i в город j на t-й итерации:

(1)

$$\begin{cases} P_{ij,k}(t) = \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in J_{i,k}} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}}, \text{если } j \in J_{i,k} \\ P_{ij,k}(t) = 0, \text{если } \ j \notin J_{i,k} \end{cases},$$

где α и β — два регулируемых параметра, задающие веса следа феромона и видимости при выборе маршрута. Если α=0, то будет выбран ближайший город, что соответствует жадному алгоритму в классической теории оптимизации. При β=0 работает только феромонное усиление, что приводит к тому, что маршруты вырождаются к одному субоптимальному решению.

Вообще говоря, правило (1) определяет лишь вероятность перехода в тот или иной город, который определяется совершенно случайным образом.

Значения вероятностей $P_{ij,k}(t)$ не изменяется во время итерации, но для разных муравьев в одном и том же городе, в общем случае, различны, т. к. $P_{ij,k}(t)$ зависит от $J_{i,k}$ — списка городов, которые муравей k еще не посетил.

После завершения маршрута каждый муравей k откладывает на ребре (i,j) такое количество феромона:

$$\Delta\tau_{ij,k}(t) = \begin{cases} \dfrac{Q}{L_k(t)}, \text{если } (i,j) \in T_k(t) \\ 0, \quad \text{если } \ (i,j) \notin T_k(t) \end{cases},$$

где $T_k(t)$ — маршрут, пройденный муравьем k на итерации t; $L_k(t)$ —длина этого маршрута; Q — регулируемый параметр, значение которого выбирают одного порядка с длиной оптимального маршрута.

Кроме того, для нахождения решения необходимо обеспечить испарение феромона — уменьшение его количества, накопившееся во время предыдущих итераций. С этой целью вводится коэффициент испарения феромона p∈[0,1]. Тогда за правило обновления феромона принимается следующее:

$$\tau_{ij}(t+1) = (1-p) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t), \tag{2}$$

где

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij,k}(t),$$

а m — количество муравьев в колонии.

В начале оптимизации количество феромона между каждым из городов берется за τ_0 — некоторую небольшую положительную величину. Количество муравьев во время выполнения алгоритма изменяться не будет. Слишком большая колония муравьев приводит к тому, что субоптимальные маршруты будут усиливаться, а если муравьев будет мало, то феромон будет быстро испаряться. Поэтому количество муравьев принимается равным количеству городов, и каждый будет начинать свой маршрут из своего города.

Особенности решения задачи поиска оптимального маршрута

Задача, которая рассматривается в нашей работе (рис. 1), имеет некоторые отличия от классической задачи коммивояжера.

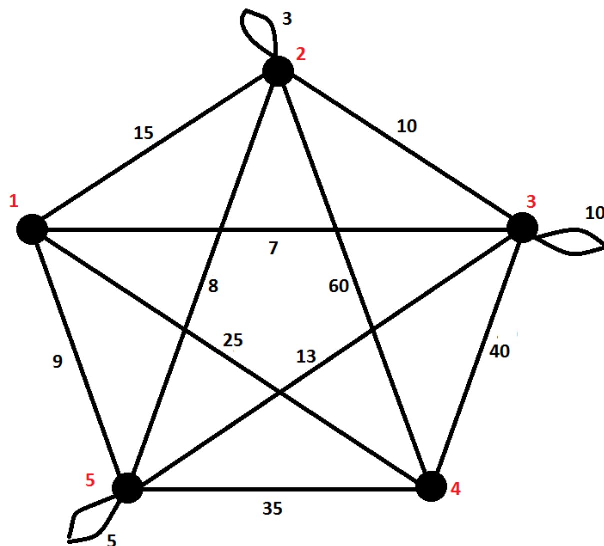

Рис. 1. Граф, отражающий задачу поиска оптимального туристического маршрута

В первую очередь, постановка нашей задачи не требует возвращения в исходную позицию: турист может начать и закончить свой путь в любой точке. Помимо того, нет необходимости совершать обход по всем вершинам, поскольку в задаче есть ограничение по времени в прохождении искомого маршрута. Указанные различия приведены на рисунках 2 и 3.

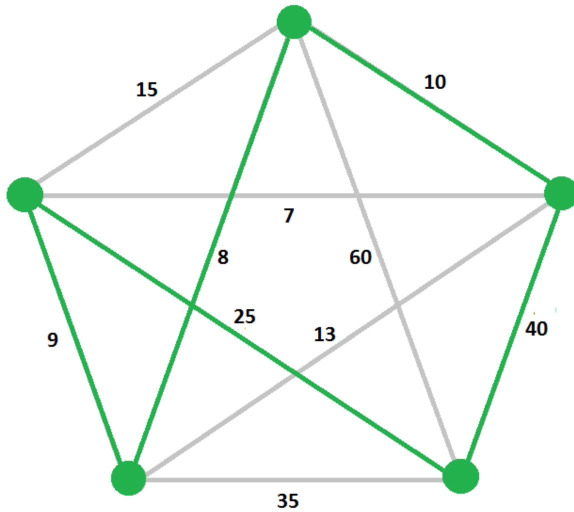

Рис. 2. Поиск маршрута в задаче коммивояжера.

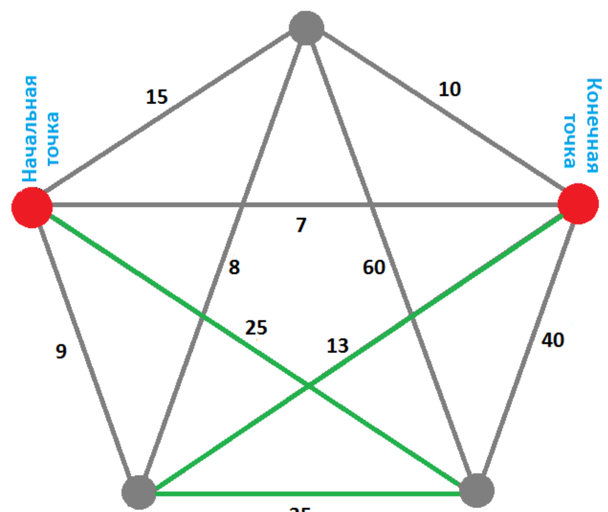

Рис. 3. Поиск маршрута в решаемой нами задаче.

Также может возникнуть необходимость выделить дополнительное время на осмотр памятников, перерыва на обед, развлечений в аквапарке и т.п. Такие ситуации необходимо рассмотреть в модели. С этой целью мы вводим дополнительные петли (под петлей понимается ребро, начало и конец которой в одной вершине [25, с. 138]) у определенных вершин графа, вес которых будет соответствовать продолжительности посещения объекта.

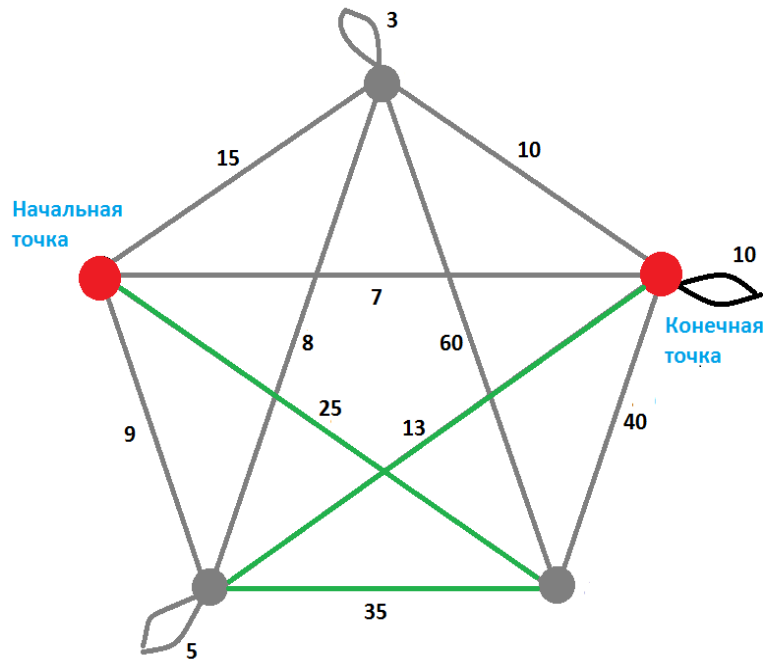

Рис. 4. Граф с добавлением петель, указывающих
время посещения объектов.

В придачу мы будем учитывать, что турист хочет посетить тот или иной объект с большим желанием, а потому предоставим ему возможность указывать такую потребность.

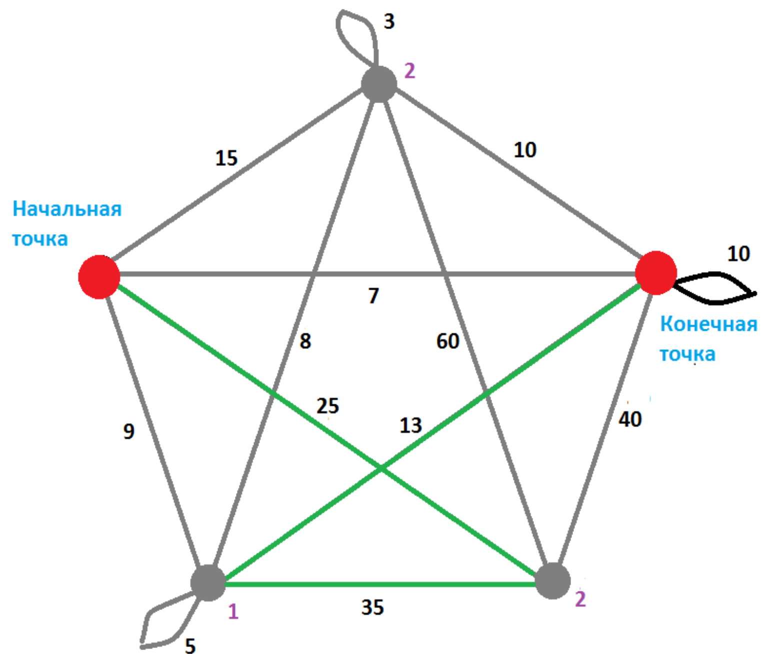

Рис. 5. Граф с указанными приоритетами

посещения вершин

В связи с имеющимися различиями между нашим случаем и задачей, рассмотренной выше, возникает необходимость модифицировать имеющиеся алгоритмы решения, чтобы они позволили нам достичь поставленной цели.

Для этого внесем ряд изменений в имеющийся муравьиный алгоритм. Поскольку начало и завершение маршрута следования задается строго определенным образом, нет смысла размещать всех муравьев в разных городах. Более оптимальным вариантом в нашем случае будет группирование колонии в начальной точке. Возможность указывать свои пожелания в плане приоритета посещения тех или иных достопримечательностей, предоставляемая туристу, приводит к необходимости изменить расчет вероятности перехода муравья из одной вершины в другую.

Так как время у туриста ограничено, будем прерывать поиск, как только планируемое время экскурсии превысит заранее заданное значение. Таким образом, мы не будем искать такого маршрута, который проходил бы через все вершины графа, как в задаче коммивояжера, но построим алгоритм так, чтобы он учитывал не только время на перемещение в конечную точку, но и приоритет вершин.

Также наш алгоритм предусматривает такие ситуации, когда на осмотр или посещение достопримечательности требуется потратить определенное время. Поэтому при поиске пути муравьи, заходящие в такую вершину, будут осуществлять переход по петле с заданным весом.

3. Программная реализация модифицированного муравьиного алгоритма

Для реализации алгоритма мы выбрали язык программирования C++ и его расширение C++/CLI для платформы .NET.

Решение поставленной задачи требует наличия информации о начальной и конечной точках маршрута, а также времени на переход между достопримечательностями. Получить необходимые сведения позволяет геоинформационный онлайн сервис «Яндекс.Карты». С этой целью в программе использовался набор свободно распространяемых кроссплатформенных библиотек GMap.NET, с открытым исходным кодом.

На следующем шаге для поиска оптимального маршрута применялся модифицированный муравьиный алгоритм, для работы которого необходимо было построить граф, представляющий схему всевозможных перемещений по городу. В качестве весов на ребрах графа использовалось время, затрачиваемое на переход от одной достопримечательности до другой. При этом граф в программе представлялся в виде двумерного массива, элементы которого, полученные из Яндекс.Карт, образовывали симметричную относительно главной диагонали матрицу. На главной диагонали располагались значения, соответствующие продолжительности посещения того или иного объекта.

Также в нашей программе использовалась матрица приоритетов, значения в которой получались из базы данных, но могли модифицироваться пользователем.

В начале работы алгоритма распределение феромона τ_0 устанавливалось на всех ребрах равным 0.5. Такое значение было выбрано в ходе большого числа экспериментов для достижения наилучших результатов.

В классическом муравьином алгоритме на вероятность перехода в другой город влияло только расстояние до него. В нашем же случае необходимо также учитывать важность посещения достопримечательностей. Для того, чтобы на вероятность перехода муравья, оба эти параметра влияли равным образом, нами производилась «нормировка» приоритетов городов по следующей формуле:

$$pr_i = \frac{pr_1}{pr_i \cdot S_{pr}}, i = \overline{1, n},$$

где n — число вершин графа, pr_i — приоритет посещения i-ой вершины, а S_{pr} — нормировочный коэффициент, вычисляемый по формуле:

$$S_{pr} = \sum_{\substack{j=1 \\ pr_j \neq pr_h}}^{n} \frac{pr_1}{pr_j}, \qquad \forall h < j, h \in \mathbb{N}.$$

Обозначим через $M_k(t)$ множество номеров вершин, посещенных k-м муравьем на t-ой итерации. Тогда оптимальным будем считать тот маршрут, для которого функция $F = \sum_{i \in M_k(t)} pr_i$ принимает наибольшее значение.

Далее начинаем запускать партии из m муравьев. Для лучших результатов их число принимается равным числу городов, т.е. m=n. Формула нахождения вероятности перехода из одного города в другой не изменяется по сравнению с классическим алгоритмом и определяется по формуле (1). При этом сам переход осуществляется по принципу «колеса рулетки» в соответствии с рассчитанными значениями вероятностей.

Так как общая длина маршрута, указываемая пользователем, ограничена, то осуществляется отбор тех путей, длины которых не превышали бы эту величину. При нахождении более длинного пути, поиск следует завершить заранее. Поскольку расчет длины маршрута необходимо выполнить с учетом того, что после осмотра всех достопримечательностей туристу необходимо вернуться в конечный город, то длина

соответствующего пути автоматически добавляется к пройденной муравьем дистанции. При наличии возможности продолжить построение маршрута, искусственно добавленное ребро графа удаляется. Кроме того, во всех вершинах необходимо осуществить переход по петле.

После выполнения каждой итерации на ребре (i,j) откладывается феромон, определяемый по формуле:

$$\Delta\tau_{ij,k}(t) = \begin{cases} \dfrac{Q \cdot \sqrt[p]{f_{k,t}}}{L_k(t) \cdot F}, \text{если } (i,j) \in T_k(t) \\ 0, \quad\quad \text{если } (i,j) \notin T_k(t) \end{cases},$$

где $T_k(t)$ — маршрут, пройденный муравьем k на итерации t; $L_k(t)$ —длина этого маршрута; $f_{k,t} = \sum_{i \in T_k(t)} pr_i$; $F = \max_{k,t} f_{k,t}$; Q — величина порядка длины оптимального маршрута, принимаемая нами равной S_{max} — значению равному указанному туристом ограничению по длине маршрута.

Поиск оптимального маршрута будем завершать после r итераций или же, когда все муравьи в рамках одной итерации пробегут по одинаковым маршрутам. В целях предотвращения случаев случайного выбора одинаковых маршрутов всеми муравьями на начальном этапе работы алгоритма, мы запускаем не менее, чем q итераций. В ходе проведенных экспериментов были получены оптимальные значения для r и q, которые составили 5n и n соответственно.

Для проверки работоспособности приложения нами были проведены его испытания на примере поиска туристического маршрута в городе Казань.

В качестве примера был выбран ряд достопримечательностей.

Начальная точка: Парк Горького. Достопримечательности: Музей 1000-летия Казани (приоритет 1, дополнительное время 30 минут), театр кукол (приоритет 3, дополнительное время 50 минут), Кул Шариф (приоритет 1, дополнительное время 20 минут). Конечная точка: площадь

Тукая. Было поставлено ограничение в 72 минуты и получен следующий результат:

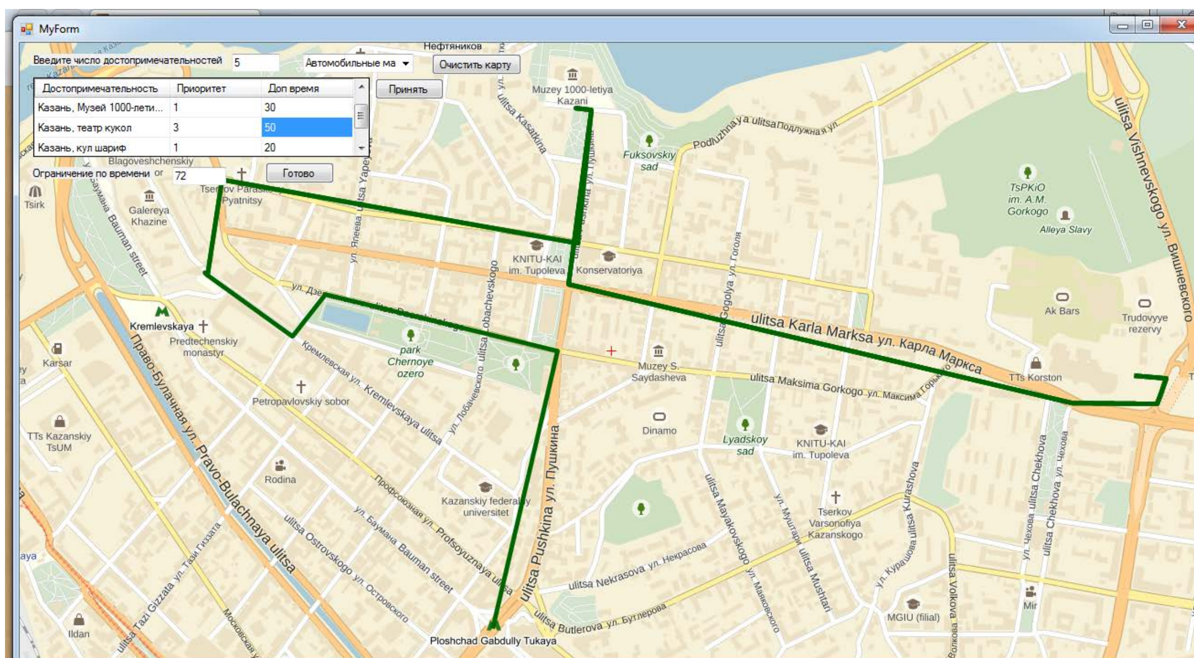

Рис.6. Результат работы при заданных ограничениях.

Указанный путь начинается с парка Горького, затем проходит через музей 1000-летия Казани и Кул Шариф и заканчивается на площади Тукая. Через театр кукол наш маршрут не проходит в связи с имеющимися ограничениями по времени, а также низким приоритетом указанной достопримечательности.

Заключение

В работе были изучены возможности геоинформационных систем при создании математической модели поиска оптимального туристического маршрута. Нами было установлено, что ГИС и, в частности "Яндекс-карты", обеспечивает удобство как пользовательского интерфейса, так и интерфейса прикладного программирования (API). Это позволило организовать интуитивно-понятное взаимодействие пользователя с приложением, а также сократить временные затраты на получение ряда исходных данных (географического расположения достопримечательностей и путей между ними, а также времени, затрачиваемого на перемещения).

Поиск оптимального маршрута математическими методами потребовал разработки соответствующей модели. В работе предложена модификация классического муравьиного алгоритма, позволяющая решить поставленную задачу с учетом всех ограничений, указанных пользователем.

В результате было разработано приложение, которое дает возможность пользователю указывать требования к построению маршрута следования, а также производит вычисления для поиска оптимального пути и наглядно представляет результаты исследования.

Таким образом, можно сказать, что природные механизмы поиска и, в частности, модификации муравьиных алгоритмов, приведенные в данной работе, являются эффективным средством решения поставленной задачи поиска оптимального туристического маршрута.

Список литературы.

1. Dorigo, M. Distrbutes Optimization by Ant Colonies / M. Dorigo, A. Colomi, V. Maniezzo // Actes de la première conférence européenne sur la vie artificielle. — 1991. — P. 134-142.
2. Dorigo, M. Optimization, Learning and Natural Algorithms / M. Dorigo // PhD thesis, Politecnico di Milano. — 1992.
3. Bianchi, L. An ant colony optimization approach to the probabilistic traveling salesman problem / L. Bianchi, L. M. Gambardella, M. Dorigo// Seventh International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science. — 2002.
4. Лореш М.А. Разработка и исследование алгоритмов муравьиной колонии для решения задач оптимального размещения предприятий: Дис. … канд. техн. наук : 05.13.01 — дата защиты 29.06.06. Омск, 2006. — 113 с.
5. Дудин, П.А. Гибридные алгоритмы муравьиной колонии для идентификации параметров нечетких систем: Дис. … канд. техн. наук : 05.13.18 — дата защиты 15.12.11. Томск, 2011. — 180 с.
6. Глушко, С.И. Иерархические нечеткие многоканальные муравьиные алгоритмы и комплекс программ оптимизации телекоммуникационных сетей нефтетранспортных предприятий: Дис. … канд. техн. наук : 05.13.18 — дата защиты 27.09.13. Москва, 2013. — 145 с.

7. Самардак, А.С. Геоинформационные системы / А.С. Самардак — Владивосток: Дальневосточный Государственный Университет, 2005. — 124 с.
8. Питенко, А.А. Нейросетевой анализ в геоинформационных системах / А.А. Питенко — Красноярск: Интеграция, 2000 — 97 с.
9. Что такое ГИС [Электронный ресурс] // Esri CIS. Геоинформационные системы. Преимущества географического подхода. URL: http://esri-cis.ru/concept_arkgisa/press/whatgis.php.
10. Геоинформатика: учеб. для студ. вузов / Е.Г. Капралов [и др.]. — М. : Академия, 2005. — 480 с.
11. Журкин, И.Г. Геоинформационные системы / И.Г. Журкин, С.В. Шайтура— СПб. : Кудиц-пресс, 2009. — 272 с.
12. Асанов, М.О., Дискретная математика: графы, матроиды, алгоритмы / М.О. Асанов, В.А. Баранский, В.В. Расин — СПб. : Лань, 2010. — 368 с.
13. Свами, М. Графы, сети и алгоритмы / М. Свами, К. Тхуласираман — М. : Мир, 1984. — 454 с.
14. Иванко, Е.Е. Эмпирический метод DropBy масштабированного решения задачи коммивояжера / Е.Е.Иванко // Сборник научн. трудов «Алгоритмы и программные средства параллельных вычислений». — Екб. : УрО РАН, 2010. — Вып. 10. — С. 3-7.

15. Сигал, И.Х. Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы / И.Х. Сигал, А.П. Иванова — М.: ФИЗМАТЛИТ, 2003. — 240 с.
16. Мудров, В. Задача о коммивояжере : учеб. пособие / В. Мудров — М.: «Знание», 1969. — 66 с.
17. Ерзин, А.И. Задачи маршрутизации: учеб. Пособие / А.И. Ерзин, Ю.А. Кочетов — Новосибирск: РИЦ НГУ, 2014. — 43с.
18. Глава 46. Задача коммивояжёра. [Электронный ресурс] // Механико-математический факультет МГУ: [сайт]. URL: http://mech.math.msu.su/~shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml#d5e27164
19. Хэмди, А. Введение в исследование операций / А. Хэмди — М. : Вильямс, 2001. — 912 с.
20. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл; пер. С.К. Ландо. — М. : Техносфера, 2004. — 368 с.
21. Штовба, С.Д. Муравьиные алгоритмы / С.Д. Штовба // ExponentaPro. — 2003. — №4. — С. 70-75.
22. Dorigo, M. Ant algorithms and stimergy / M. Dorigo, E. Bonabeau, G. Theraulaz // Future Generation Computer Systems. — 2000. — №16. — P. 851-871.
23. Dorigo, M. Ant colony optimization / M. Dorigo, T. Stützle — A Bradford book. — 2004 — P. 321.

24. Курейчик, В.М. Гибридный алгоритм разбиения на основе природных механизмов принятия решений / В.М. Курейчик, Б.К. Лебедев, О.Б. Лебедев // Искусственный интеллект и принятие решений. — 2012. — №2. — С. 3-15.
25. Клауди, А. Карты метро и нейронные сети. Теория графов / А.Клауди. — М. : Де Агостини, 2014. — 144 с.

Приложения

MyForm.h

```
#include "ant_algoritm.h"

#pragma endregion
	private: System::Void MyForm_Load(System::Object^  sender, System::EventArgs^  e)
{
	//Настройки для компонента GMap.
	 gMapControl1->Bearing = 0;

	 //CanDragMap - Если параметр установлен в True, пользователь может
	 //перетаскивать карту с помощью правой кнопки мыши.
	 gMapControl1->CanDragMap = true;

	//Указываем, что перетаскивание карты осуществляется   использованием левой
	//клавишей мыши. По умолчанию - правая.
	 gMapControl1->DragButton = Windows::Forms::MouseButtons::Left;

	 gMapControl1->GrayScaleMode = true;

	 //MarkersEnabled - Если параметр установлен в True,  маркеры,
	 // заданные вручную будет показаны. Если нет, они не появятся.
	 gMapControl1->MarkersEnabled = true;

	 //Указываем значение максимального приближения.
	 gMapControl1->MaxZoom = 25;

	 //Указываем значение минимального приближения.
	 gMapControl1->MinZoom = 2;

	 //Устанавливаем центр приближения/удаления курсор мыши.
	 gMapControl1->MouseWheelZoomType =
	 GMap::NET::MouseWheelZoomType::MousePositionAndCenter;

	 //Отказываемся от негативного режима.
	 gMapControl1->NegativeMode = false;

	 //Разрешаем полигоны.
	 gMapControl1->PolygonsEnabled = true;

	 //Разрешаем маршруты
	 gMapControl1->RoutesEnabled = true;

	 //Скрываем внешнюю сетку карты с заголовками.
	 gMapControl1->ShowTileGridLines = false;

	 //Указываем, что при загрузке карты будет использоваться
	 //15ти кратное приближение.
	 gMapControl1->Zoom = 15;

	 //Указываем что все края элемента управления закрепляются у краев
	 //содержащего его элемента управления(главной формы),
	 //а их размеры изменяются соответствующим образом.
	 gMapControl1->Dock = Windows::Forms::DockStyle::Fill;

	 //Указываем что будем использовать карты Yandex.
	 gMapControl1->MapProvider =  GMap::NET::MapProviders::GMapProviders::YandexMap;
	 GMap::NET::GMaps::Instance->Mode = GMap::NET::AccessMode::ServerOnly;
```

```
        /*  //Если вы используете интернет через прокси сервер,
        //указываем свои учетные данные.
        GMap::NET::MapProviders::GMapProvider::WebProxy =
               System::Net::WebRequest::GetSystemWebProxy();
        GMap::NET::MapProviders::GMapProvider::WebProxy->Credentials =
               System::Net::CredentialCache::DefaultCredentials; */

        gMapControl1->Position = GMap::NET::PointLatLng(55.795308, 49.106301);


         DataTable^ dtRouter = gcnew DataTable();
        //Добавляем в инициализированную таблицу, новые колонки.
         dtRouter->Columns->Add("Шаг");
         dtRouter->Columns->Add("Нач. точка (latitude)");
         dtRouter->Columns->Add("Нач. точка (longitude)");
         dtRouter->Columns->Add("Кон. точка (latitude)");
         dtRouter->Columns->Add("Кон. точка (longitude)");
         dtRouter->Columns->Add("Время пути");
         dtRouter->Columns->Add("Расстояние");
         dtRouter->Columns->Add("Описание маршрута");

         //Добавляем способы перемещения.
         comboBox1->Items->Add("Автомобильные маршруты");
         comboBox1->Items->Add("Пешеходные маршруты");
         comboBox1->Items->Add("Велосипедные маршруты");
         comboBox1->Items->Add("Маршруты общественного транспорта");

         //Выставляем по умолчанию способ перемещения:
         //Автомобильные маршруты по улично-дорожной сети.
         comboBox1->SelectedIndex = 0;

         //GMap::NET::LanguageType::Russian;

        }

DataTable dtRouter;


public: String^ HtmlToPlainText(String^ html)
         {
                html = html->Replace("/b", "");
                return html->Replace("b", "");
         }

value struct router{
       int i;
       String^ sLat;
       String^ sLng;
       String^ eLat;
       String^ eLng;
       String^ duration;
       String^ distance;
       String^ instr;
};

private: System::Void izm_ch_dost(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e) {
       if ((e->KeyChar < 48 || e->KeyChar > 57) && e->KeyChar != '\b')
               e->Handled = true;
}


private: System::Void Accept_Click(System::Object^  sender, System::EventArgs^  e) {
        try{
```

```
			n = Convert::ToInt32(chislo_dost->Text);

			Matrica->Columns->Clear();  //удаление старых столбцов
			Matrica->Rows->Clear();  //удаление старых строк

			//добавление 3х столбцов
			Matrica->Columns->Add("Достопримечательность","Достопримечательность");
			Matrica->Columns->Add("Приоритет", "Приоритет");
			Matrica->Columns->Add("Доп время", "Доп время");

			for (int i = 0; i < n; i++){  //добавление n строк
				Matrica->Rows->Add("", "");
			}
			for (int i = 0; i<n; i++){
				Matrica->Rows[i]->Cells[1]->Value = 1;
			}
			for (int i = 0; i<n; i++){
				Matrica->Rows[i]->Cells[2]->Value = 0;
			}

		}
		catch (...){
			MessageBox::Show("Кажется, что-то пошло не так!");
		}
	}
	private: System::Void Accept_dost_Click(System::Object^  sender, System::EventArgs^  e) {

		try{
			List<router>^ dtRouter = gcnew List<router>();
			dtRouter->Clear();
			//Создаем. список способов перемещения.
			List <String^> mode;
			//Автомобильные маршруты по улично-дорожной сети.
			mode.Add("driving");
			//Пешеходные маршруты по прогулочным дорожкам и тротуарам.
			mode.Add("walking");
			//Велосипедные маршруты по велосипедным дорожкам и предпочитаемым улицам.
			mode.Add("bicycling");
			//Маршруты общественного транспорта.
			mode.Add("transit");

			int Smax = Convert::ToDouble(Maximum->Text);
			Double*pr = new Double[n];
			for (int i = 0; i < n; i++)
				pr[i] = Convert::ToDouble(Matrica->Rows[i]->Cells[1]->Value);
			Double**D = new Double*[n];
			for (int i = 0; i < n; i++)
				D[i] = new Double[n];

			for (int i = 0; i < n; i++){
			      for (int j = 0; j < n; j++){
					D[i][j] = 0;
				}
			}

			for (int i = 0; i < n; i++)
				D[i][i] = Convert::ToDouble(Matrica->Rows[i]->Cells[2]->Value);


			for (int i = 0; i < n; i++){
				for (int j = 0; j < i; j++){
			String^ nt = Convert::ToString(Matrica->Rows[i]->Cells[0]->Value);
			String^ kt = Convert::ToString(Matrica->Rows[j]->Cells[0]->Value);
				//Фомируем запрос к API маршрутов Google.
```

```
        String^ url = String::Format(
"http://maps.googleapis.com/maps/api/directions/xml?origin={0},&destination={1}&sensor=fa
lse&language=ru&mode={2}", Uri::EscapeDataString(nt), Uri::EscapeDataString(kt),
Uri::EscapeDataString(mode[comboBox1->SelectedIndex]));
                      //Выполняем запрос к универсальному коду ресурса (URI).
        System::Net::HttpWebRequest^ request =
dynamic_cast<System::Net::HttpWebRequest^>(System::Net::WebRequest::Create(url));

        //Получаем ответ от интернет-ресурса.
        System::Net::WebResponse^ response = request->GetResponse();

        //Экземпляр класса System.IO.Stream чтения данных из интернет-ресурса.
        System::IO::Stream^ dataStream = response->GetResponseStream();

        //Инициализируем новый экземпляр класса
        //System.IO.StreamReader для указанного потока.
        System::IO::StreamReader^ sreader = gcnew System::IO::StreamReader(dataStream);

        //Считываем поток от текущего положения до конца.
        String^ responsereader = sreader->ReadToEnd();

        //Закрываем поток ответа.
        response->Close();

        System::Xml::XmlDocument^ xmldoc = gcnew System::Xml::XmlDocument();

        xmldoc->LoadXml(responsereader);

        if (xmldoc->GetElementsByTagName("status")[0]->ChildNodes[0]->InnerText == "OK"){
               System::Xml::XmlNodeList^ nodes = xmldoc->SelectNodes("//leg//step");

        String^ vremya = xmldoc->GetElementsByTagName("duration")
              [nodes->Count]->ChildNodes[1]->InnerText;

       vremya=vremya->Remove(vremya->IndexOf(" "),vremya->Length - vremya->IndexOf(" "));
        D[i][j] = Convert::ToDouble(vremya);
        }

        // Создаем новый список маркеров, с указанием компонента
        //в котором они будут использоваться и названием списка.
        GMap::NET::WindowsForms::GMapOverlay^ markersOverlay =
              gcnew Map::NET::WindowsForms::GMapOverlay();
       //Очищаем все маршруты.
       markersOverlay->Routes->Clear();
       //Обновляем карту.
       gMapControl1->Refresh();
              }
       }

       for (int i = 0; i < n; i++){
               for (int j = i; j < n; j++){
                      D[i][j] = D[j][i];
               }
        }
        double*spg = new double[n];
        for (int i = 0; i < n; i++)
               spg[i] = 0;
        //применение муравьиных алгоритмов
        ant_algoritm aa(n, D, Smax, pr);
        aa.algoritm(n, D, Smax, pr,spg);

        int chpg = 0;
        for (int i = 0; i < n; i++){
               if (spg[i]>0){ chpg++; } }
```

```
        int*otvet = new int[chpg+1];
        otvet[0] = 0;
        for (int i = 0; i < chpg; i++){
               otvet[i + 1] = spg[i];
        }


        //построение на карте
        for (int i = 0; i < chpg; i++){
        String^ nt = Convert::ToString(Matrica->Rows[otvet[i]]->Cells[0]->Value);
        String^ kt = Convert::ToString(Matrica->Rows[otvet[i+1]]->Cells[0]->Value);
        // Фомируем запрос к API маршрутов Google.
        String^ url = String::Format(
"http://maps.googleapis.com/maps/api/directions/xml?origin={0},&destination={1}&sensor=fa
lse&language=ru&mode={2}", Uri::EscapeDataString(nt), Uri::EscapeDataString(kt),
Uri::EscapeDataString(mode[comboBox1->SelectedIndex]));

        //Выполняем запрос к универсальному коду ресурса (URI).
       System::Net::HttpWebRequest^ request =
       dynamic_cast<System::Net::HttpWebRequest^>(System::Net::WebRequest::Create(url));

        //Получаем ответ от интернет-ресурса.
        System::Net::WebResponse^ response = request->GetResponse();

        //Экземпляр класса System.IO.Stream для чтения данных из интернет-ресурса.
        System::IO::Stream^ dataStream = response->GetResponseStream();

        //Инициализируем новый экземпляр класса
        //System.IO.StreamReader для указанного потока.
        System::IO::StreamReader^ sreader = gcnew System::IO::StreamReader(dataStream);

        //Считываем поток от текущего положения до конца.
       String^ responsereader = sreader->ReadToEnd();

        //Закрываем поток ответа.
        response->Close();

        System::Xml::XmlDocument^ xmldoc = gcnew System::Xml::XmlDocument();
        xmldoc->LoadXml(responsereader);
if (xmldoc->GetElementsByTagName("status")[0]->ChildNodes[0]->InnerText == "OK") {
        System::Xml::XmlNodeList^ nodes = xmldoc->SelectNodes("//leg//step");

        //Формируем строку для добавления в таблицу.
        router dr;// = gcnew array <Object^>(8);
        for (int i = 0; i < nodes->Count; i++){
        //Указываем что массив будет состоять из восьми значений.

        //Номер шага.
        dr.i = i;
        //Получение координат начала отрезка.
        dr.sLat = xmldoc->SelectNodes("//start_location")->Item(i)->
              SelectNodes("lat")->Item(0)->InnerText->ToString();
        dr.sLng = xmldoc->SelectNodes("//start_location")->Item(i)->
              SelectNodes("lng")->Item(0)->InnerText->ToString();
        //Получение координат конца отрезка.
        dr.eLat = xmldoc->SelectNodes("//end_location")->Item(i)->
              SelectNodes("lat")->Item(0)->InnerText->ToString();
        dr.eLng = xmldoc->SelectNodes("//end_location")->Item(i)->
              SelectNodes("lng")->Item(0)->InnerText->ToString();
        //Получение времени необходимого для прохождения этого отрезка.
        dr.duration = xmldoc->SelectNodes("//duration")->Item(i)->
              SelectNodes("text")->Item(0)->InnerText->ToString();
        //Получение расстояния, охватываемое этим отрезком.
        dr.distance = xmldoc->SelectNodes("//distance")->Item(i)->
```

```
        SelectNodes("text")->Item(0)->InnerText->ToString();
    //Получение инструкций для этого шага, представл. в виде текстовой строки  HTML.
    dr.instr = HtmlToPlainText(xmldoc->SelectNodes("//html_instructions")->
        Item(i)->InnerText->ToString());
    //Добавление шага в таблицу.
    dtRouter->Add(dr);
    }

    //Переменные для хранения координат начала и конца пути.
    double latStart = 0.0;
    double lngStart = 0.0;
    double latEnd = 0.0;
    double lngEnd = 0.0;

    //Получение координат начала пути.
    latStart = System::Xml::XmlConvert::ToDouble(xmldoc->
    GetElementsByTagName("start_location")[nodes->Count]->ChildNodes[0]->InnerText);
    lngStart = System::Xml::XmlConvert::ToDouble(xmldoc->
    GetElementsByTagName("start_location")[nodes->Count]->ChildNodes[1]->InnerText);
    //Получение координат конечной точки.
    latEnd = System::Xml::XmlConvert::ToDouble(xmldoc->
    GetElementsByTagName("end_location")[nodes->Count]->ChildNodes[0]->InnerText);
    lngEnd = System::Xml::XmlConvert::ToDouble(xmldoc->
    GetElementsByTagName("end_location")[nodes->Count]->ChildNodes[1]->InnerText);

    // Создаем новый список маркеров, с указанием компонента
    //в котором они будут использоваться и названием списка.
    GMap::NET::WindowsForms::GMapOverlay^ markersOverlay =
         gcnew GMap::NET::WindowsForms::GMapOverlay();
   //Инициализация нового ЗЕЛЕНОГО маркера, с указанием координат начала пути.

GMap::NET::WindowsForms::Markers::GMarkerGoogle^ markerG =
       gcnew GMap::NET::WindowsForms::Markers::GMarkerGoogle(
       GMap::NET::PointLatLng(latStart, lngStart),
       GMap::NET::WindowsForms::Markers::GMarkerGoogleType::blue);

 markerG->ToolTip = gcnew GMap::NET::WindowsForms::ToolTips::GMapRoundedToolTip(markerG);
    //Указываем, что подсказку маркера, необходимо отображать всегда.
    markerG->ToolTipMode = GMap::NET::WindowsForms::MarkerTooltipMode::Always;

    //Инициализация нового Красного маркера, с указанием координат конца пути.

GMap::NET::WindowsForms::Markers::GMarkerGoogle^ markerR =
    gcnew GMap::NET::WindowsForms::Markers::GMarkerGoogle(
         GMap::NET::PointLatLng(latEnd, lngEnd),
        GMap::NET::WindowsForms::Markers::GMarkerGoogleType::blue);
 markerG->ToolTip = gcnew GMap::NET::WindowsForms::ToolTips::GMapRoundedToolTip(markerG);

    //Указываем, что подсказку маркера, необходимо отображать всегда.
    markerR->ToolTipMode = GMap::NET::WindowsForms::MarkerTooltipMode::Always;

    //Создаем список контрольных точек для прокладки маршрута.
    List <PointLatLng>^ list = gcnew List<PointLatLng>();

    //Проходимся по определенным столбцам для получения
    //координат контрольных точек маршрута и занесением их в список координат.
    for (int i = 0; i < dtRouter->Count; i++){
        Double dbStartLat = Double::Parse(dtRouter[i].sLat,
        System::Globalization::CultureInfo::InvariantCulture);
         Double dbStartLng = Double::Parse(dtRouter[i].sLng,
        System::Globalization::CultureInfo::InvariantCulture);

         list->Add(PointLatLng(dbStartLat, dbStartLng));
```

```
            double dbEndLat = double::Parse(dtRouter[i].eLat,
            System::Globalization::CultureInfo::InvariantCulture);
             double dbEndLng = double::Parse(dtRouter[i].eLng,
            System::Globalization::CultureInfo::InvariantCulture);

            list->Add(PointLatLng(dbEndLat, dbEndLng));
}

        // Создаем маршрут на основе списка контрольных точек.

       GMap::NET::WindowsForms::GMapRoute^ r = gcnew GMap::NET::WindowsForms::
            GMapRoute(list, "Route");

        //Указываем, что данный маршрут должен отображаться.
        r->IsVisible = true;
        //Устанавливаем цвет маршрута.
        r->Stroke->Color = Color::DarkGreen;
        //Добавляем маршрут.
        markersOverlay->Routes->Add(r);

        //Добавляем в компонент, список маркеров и маршрутов.
        gMapControl1->Overlays->Add(markersOverlay);

        //Указываем, что при загрузке карты будет использоваться 15ти кратное прибл.
        gMapControl1->Zoom = 15;
    }
  }
}
catch (...){
        MessageBox::Show("Кажется, что-то пошло не так!");}

}

private: System::Void Clear_button_Click(System::Object^  sender, System::EventArgs^ e) {
       GMap::NET::WindowsForms::GMapOverlay^ markersOverlay =
             gcnew GMap::NET::WindowsForms::GMapOverlay();
                    //Очищаем все маршруты.
                    markersOverlay->Routes->Clear();
                    gMapControl1->Overlays->Clear();
                    gMapControl1->Refresh();
}
};
}
```

ant_algoritm.h

```
#pragma once
class ant_algoritm
{
public:
       ant_algoritm(int n, double **D, double Smax, double *&pr/*, double*&spg*/);
       ~ant_algoritm();
protected:
       int n;//кол-во городов
       int m;//кол-во муравьев
       double** D;//матрица расстояний между городами
       double Smax;//ограничение
       double *pr;//матрица приоритетов
       double** tau;//феромон
       double*pg;//список посещенных городов: 0-не посетил, 1-посетил, 2-нах-ся в
                  //этом городе; первый индекс-номер муравья, второй-город
       double** dtau;
       double Q;
       const double tau0 = 0.5;//начальное значение феромона на ребрах
```

```
	const double alpha = 0.8;//константа
	const double beta = 0.3;//константа
	const double p = 0.2;//коэф испарения
	double *P;//вероятность перехода муравья в другой город
	double L;//длина пройденного муравьем пути
	double*Lm;//столько прошел m-ый муравей
	double** pgm;//посещенные муравьем города

public:
	void nach_feromon(int n, double**&tau);//начальные феромоны
	void nach_dannie(int n, double*&pr, double*&pg);//нормировочные коэфф для
	//приоритетов, и список посещ городов
	void veroyatn_perehod(int n, double**D, double**&P, double*pg, double**tau,
		double*pr);//вычисл вероятности перехода муравья в следующий город
	void perehod(int n, double**D, double**P, double*&pg, double &L, int*&spg_vr,
		double &func, int q, double*pr);//переход муравья в следующий город
	void izm_feromon(int m, double**&tau, int**pgm, double*Lm, double Smax, double F,
	    double*funcm);//изменение феромона после того как муравей прошел весь маршрут
	void algoritm(int n, double **D, double Smax, double *pr, double*spg);
		//сам алгоритм, все должен объединить
	void undo(int n, double**D, double*&pg, double &L, int*&spg_vr, double &func,
	      int q, double*pr);//отменяет изменения из перехода
	void create(int n, int*&matr);
	void create(int n, int**&matr);
	int chislo_povtor(int n, int**pgm);
};
```

ant_algoritm.cpp

```
#include "ant_algoritm.h"
#include <iostream>
#include <math.h>
#include <time.h>
using namespace std;

ant_algoritm::ant_algoritm(int n, double **D, double Smax, double *&pr/*,double*&spg*/)
{
	this->n = n;
	this->D = D;
	this->Smax = Smax;
	this->pr = pr;
	//this->spg = spg;
}


ant_algoritm::~ant_algoritm()
{
}


void ant_algoritm::nach_feromon(int n, double** &tau)
	//начальные феромоны в городах равны константе
{
	for (int i = 0; i<n; i++){
		for (int j = 0; j<n; j++){
			if (i != j){
				tau[i][j] = tau0;
			}
			else{
				tau[i][j] = 0;
			}
		}
	}
}
```

```
void ant_algoritm::nach_dannie(int n, double*&pr, double*&pg){
	//обнуляем списки посещенных городов для всех муравьев
	for (int i = 0; i < n; i++){
		if (i != 0){ pg[i] = 0; }
		else{ pg[i] = 2; }
	}

	//нормирование приоритетов
	for (int i = 0; i < n; i++){
		pr[i] = pr[0] / pr[i];
	}

	double Spr = pr[0];//делитель

	double sh = 0;//счетчик

	//ищем этот делитель
	for (int i = 1; i < n; i++){
		for (int j = 0; j < i; j++){
			if (sh<1) if (pr[i]<pr[j] || pr[i]>pr[j]){ Spr = Spr + pr[i]; sh++; }
		}  sh = 0;
	}

	//окончательное нормирование приоритетов
	for (int i = 0; i < n; i++){
		pr[i] = pr[i] / Spr;
	}

}


void ant_algoritm::veroyatn_perehod(int n, double**D, double**&P, double*pg, double**tau,
			double*pr){

	int n1 = -1;//город из которого мы выходим
	//ищем этот город
	for (int i = 0; i < n; i++){
		if (pg[i]>1){ n1 = i; }
	}

	double*znam = new double[n];//знаменатель
	for (int i = 0; i < n; i++) znam[i] = 0;

	for (int i = 0; i < n; i++){
		for (int j = 0; j < n; j++){
		if (pg[j] < 1){
		if (i != j)znam[i] = znam[i] + pow(tau[i][j], alpha)*pow(1 / D[i][j], beta;
			}
		}
	}

	//собственно вычисление вероятности перехода в город i
	for (int i = 0; i < n; i++){
		for (int j = 0; j < n; j++){
		if (pg[j] < 1){ if (i != j)
			P[i][j] = 100*pow(tau[i][j], alpha)*pow(1 / D[i][j], beta/ znam[i];
		}
		else{ P[i][j] = 0; }
		}
	}

}
```

```
void ant_algoritm::perehod(int n, double**D, double**P, double*&pg, double &L,
	int*&spg_vr, double &func, int q, double*pr){
	//число на рулетке

	double a;
	a = rand() % 99;

	int n1 = -1;//город из которого мы выходим
	//ищем этот город
	for (int i = 0; i < n; i++){
		if (pg[i]>1){ n1 = i; }
	}

	int gorj = 0;//город в который мы пойдем
	int sh = 0;//счетчик
	while (a >= 0 && gorj<n){
		if (pg[gorj]<1){
			a = a - P[n1][gorj];
		}gorj++;
	}
	gorj--;


	pg[n1] = 1;
	pg[gorj] = 2;

	if (n1 > 0 && gorj<4){
		L = L + D[n1][gorj] + D[gorj][gorj] + D[gorj][n - 1] - D[n1][n - 1];
	}
	if (n1 < 1 && gorj<4){ L = L + D[n1][gorj] + D[gorj][gorj] + D[gorj][n - 1]; }
	if (n1<1 && gorj>3){ L = L + D[n1][gorj]; }
	spg_vr[q] = gorj;
	func = func + pr[gorj] + 1;
}



void ant_algoritm::izm_feromon(int m, double**&tau, int**pgm, double*Lm, double Smax,
double F, double*funcm){

	double**dtau = new double*[n];//в формуле изменения феромона
	for (int i = 0; i<n; i++)
		dtau[i] = new double[n];

	for (int i = 0; i < n; i++){
		for (int j = 0; j < n; j++){
			dtau[i][j] = 0;
		}
	}


	for (int i = 0; i < n; i++){
		for (int j = 0; j < n; j++){
		{ if (pgm[i][j + 1]>0){
dtau[(pgm[i][j])][(pgm[i][j + 1])] = (Smax*pow(F, -1)) / (Lm[i] * pow(funcm[i], -1./ p));
		} }
	}
}

	for (int i = 0; i < n; i++){
	{ if (pgm[i][0]>0){
	dtau[0][pgm[i][0]] = (Smax*pow(F, -1)) / (Lm[i] * pow(funcm[i], -1. / p; } }
```

```
        }

        for (int i = 0; i < n; i++){
                for (int j = 0; j < i; j++){
                        tau[i][j] = (1 - p)*tau[i][j] + dtau[i][j] + dtau[j][i];
                }
        }
        for (int i = 0; i < n; i++){
                for (int j = 0; j < i; j++){
                        tau[j][i] = tau[i][j];
                }
        }
}



void ant_algoritm::algoritm(int n, double **D, double Smax, double *pr, double*spg){

        double** tau = new double*[n];//феромон
        for (int i = 0; i<n; i++)
                tau[i] = new double[n];

        double*pg = new double[n];//список посещенных городов: 0-не посетил, 1-посетил,
                //2-  нах-ся в этом городе; первый индекс-номер муравья, второй-город

        double **P = new double*[n];//вероятность перехода муравья в другой город
        for (int i = 0; i<n; i++)
                P[i] = new double[n];

        double L = 0;//длина маршрута

        for (int i = 0; i < n; i++)
                spg[i] = 0;

        int* spg_vr = NULL;
        create(n, spg_vr);


        double func = 0;//максимизируемая функция
        double F = 0;//окончательное значение макс функции
        int q = 0;//счетчик
        int v = 0;//счетчик
        int m = n;//число муравьев

        double*Lm = new double[m];

        int** pgm = NULL;//посещенные муравьем города
        create(n, pgm);


        nach_feromon(n, tau);
        nach_dannie(n, pr, pg);
        double lv = 0;
        double*funcm = new double[m];//максимиз функция для m муравья

        int povtor = 0;

        for (int iter = 0; iter < 5 * n && (povtor<n || iter<n); iter++){
                veroyatn_perehod(n, D, P, pg, tau, pr);
                for (int i = 0; i < m; i++){
                        int qv = 0;
                        while (L <= Smax && v < n-1 && spg_vr[q - 1]<n-1 && qv<4 * n){
                                perehod(n, D, P, pg, L, spg_vr, func, q, pr);
                                if (L > Smax){
```

```
                                        undo(n, D, pg, L, spg_vr, func, q, pr); v--; q--;
                                }
                                else{ Lm[i] = L; }
                                v++; q++; qv++;
                        }funcm[i] = func;
                        //если нашлась максимизирующая функция побольше
                        if (F <= func){
                                F = func; lv = L;
                                for (int j = 0; j < n; j++)
                                        spg[j] = spg_vr[j];
                        }


                        //запоминаем список посещ городов для i-го муравья
                        for (int j = 0; j < m; j++){
                                pgm[i][j] = spg_vr[j];
                        }


                        L = 0; q = 0; v = 0; func = 0;
                        pg[0] = 2;
                        for (int j = 1; j < n; j++){
                                pg[j] = 0;
                        }

                        for (int j = 0; j < n; j++)
                                spg_vr[j] = 0;
                }

                povtor = chislo_povtor(n, pgm);

                izm_feromon(m, tau, pgm, Lm, Smax, F, funcm);
                create(n, pgm);
        }
}


void ant_algoritm::undo(int n, double**D, double*&pg, double &L, int*&spg_vr, double
&func, int q, double*pr){
        func = func - pr[spg_vr[q]] - 1;
        if (q>0){
        if (spg_vr[q - 1] > 0 && spg_vr[q] < 4){
        L = L - D[spg_vr[q - 1]][spg_vr[q]] - D[spg_vr[q]][spg_vr[q]] –
                    –  D[spg_vr[q]][n - 1] + D[spg_vr[q - 1]][n - 1];
        }
        if (spg_vr[q - 1] < 1 && spg_vr[q] < 4){
    L = L - D[spg_vr[q - 1]][spg_vr[q]] - D[spg_vr[q]][spg_vr[q]] - D[spg_vr[q]][n - 1]; }
        if (spg_vr[q - 1]<1 && spg_vr[q]>3){ L = L - D[spg_vr[q - 1]][spg_vr[q]]; }
        pg[spg_vr[q - 1]] = 2;
        pg[spg_vr[q]] = 0;
        spg_vr[q] = 0;
        }
        else{ pg[spg_vr[q]] = 0; }
}


void ant_algoritm::create(int n, int*&matr){
        if (matr != NULL){ delete[] matr; }
        matr = new int[n];
        for (int i = 0; i < n; i++)
                matr[i] = 0;
}

void ant_algoritm::create(int n, int**&matr){
```

```
	if (matr != NULL){
		for (int i = 0; i < n; i++)
			delete[] matr[i];
		delete[] matr;
	}
	matr = new int*[n];
	for (int i = 0; i < n; i++){
		matr[i] = new int[n];
		for (int j = 0; j < n; j++){
			matr[i][j] = 0;
		}
	}
}


int ant_algoritm::chislo_povtor(int n, int**pgm){
	int v = 1;
	int q = 0;
	for (int i = 0; i < n - 1; i++){
		for (int j = 0; j < n; j++){
			if (pgm[i][j] == pgm[i + 1][j]){
				q++;
			}
		}
		if (q == n){
			v++;
		}
		q = 0;
	}
	return v;
}
```