Муравьиные и генетические алгоритмы для решения задачи коммивояжера

Чубенко Полина[1,□], Савичев Дмитрий[1]

Аннотация

Постановка задачи: провести анализ существующих мета-эвристических алгоритмов (муравьиных и генетических) и сравнить их трудоемкость при решении задачи коммивояжера.
Актуальность работы: задача коммивояжера является **NP**-трудной. Следовательно, решение за полиномиальное время у данной задачи отсутствует, а переборные алгоритмы крайне неэффективны. Однако существуют различные мета-эвристические алгоритмы, которые позволяют найти близкое к оптимальному решение данной задачи за приемлемое время. Необходимо выработать "критерий" применимости того или иного алгоритма при решении задачи коммивояжера при различном количестве вершин графа и имеющемся ресурсе времени для проведения расчета. Это позволит более эффективно применять различные алгоритмы для решения задач в области логистики, в которых требуется рассчитать маршрут между взаимосвязанными вершинами графа.
Цели: определение времени решения задачи о коммивояжере каждым из описанных в данной статье алгоритмов и сравнение полученных результатов, нахождение оптимальных параметров в зависимости от входных данных и описание рекомендации по применению того или иного алгоритма.
Результаты: описана рекомендация по выбору алгоритма для расчета оптимального маршрута, который зависит от количества городов в задаче коммивояжера.
Практическая значимость: состоит в возможности на основании описанной рекомендации применять более рациональный с точки зрения временных затрат алгоритм при решении задачи коммивояжера в различных областях человеческой деятельности.

Исходный код

Ключевые слова
задача коммивояжера (TravellingSalesmanProblem), комбинаторная оптимизация, муравьиные алгоритмы (AntColonyOptimization), генетические алгоритмы (GeneticAlgorithms), мета-эвристики

[1] *Физтех-школа прикладной математики и информатики, Московский физико-технический институт (национальный исследовательский университет), Москва, Россия*
[□] Автор для переписки: chubenko.pn@phystech.edu

Содержание

Введение

В последние два десятилетия при решении **NP**-трудных задач комбинаторной оптимизации исследователи все чаще применяют алгоритмы, основанные на законах природы. Среди них выделяются эволюционные алгоритмы[1], использующие математические методы, в которых заложены принципы природных механизмов принятия решений. Например, они моделируют поведение определенных видов животных, а также используют идеи из процесса эволюции.

Основная цель разработки таких методологий состоит в том, чтобы с помощью некоторого итерационного процесса сравнительно быстро находить решения близкие к оптимальному.

Среди различных алгоритмических методов, разработанных за последние годы для труднорешаемых задач дискретной оптимизации, числятся:

1. Муравьиные алгоритмы – моделируют поведение му-

[1] они принадлежат научному направлению "Природные вычисления" (англ. Natural Computing)

равьиной колонии
2. Генетические алгоритмы – моделируют естественный отбор в генетике

Задача или проблема коммивояжера (англ. Travelling Salesman Problem, TSP [1]), является одной из наиболее изученных задач комбинаторной оптимизации. В течение многих лет ее решали различными точными методами, эвристиками, адаптированными к TSP, в том числе так называемыми мета-эвристиками, к которым относятся муравьиные и генетические алгоритмы, обсуждаемые в этой статье.

Статья состоит из четырех частей: в первой части описывается постановка задачи коммивояжера; во второй и третьей частях излагается теория муравьиных и генетических алгоритмов соответственно, и описываются решения этими алгоритмами задачи TSP, также делается анализ эффективности работы алгоритмов в зависимости от параметров и начальных условий; в четвертой части рассматривается сравнение рассмотренных алгоритмов и выделение преимуществ и недостатков обоих подходов для решения задачи коммивояжера.

1. Постановка задачи коммивояжера

Задача коммивояжера была сформулирована в 1930 году на математическом коллоквиуме в Вене и является представителем задачи комбинаторной оптимизации. В классической постановке TSP продавцу выдается карта городов, и он должен посетить все города только один раз и завершить тур таким образом, чтобы длина маршрута была наикратчайшей из возможных. Формализуя, нам дан набор городов $cities = \{c_1, c_2, \ldots, c_N\}$ и для каждой пары городов указано расстояние $distance = dist(c_i, c_j)$.

Гамильтонов цикл – цикл (замкнутый путь), который проходит через каждую вершину данного графа ровно по одному разу. Другими словами, это простой цикл, в который входят все вершины графа.

Наша цель: найти гамильтонов цикл, то есть перестановку π городов, которая минимизирует

$$D = \sum_{i=1}^{N-1} dist(c_{\pi(i)}, c_{\pi(i+1)}) + dist(c_{\pi(N)}, c_{\pi(1)}) \quad (1)$$

Эта величина является длиной маршрута, которую продавец преодолел бы, посещая города в порядке, указанном перестановкой, возвращаясь в исходный город. В этой статье мы будем рассматривать метрический TSP, также известный как Δ-TSP, в котором, междугородние расстояния удовлетворяют неравенству треугольника:

$$dist(c_i, c_j) \leqslant dist(c_i, c_k) + dist(c_k, c_j)$$

Так как мы будем рассматривать города как точки в $\mathbb{R}^2$, то в качестве функции расстояния между городами будем использовать евклидову метрику. Таким образом, в условиях поставленной нами задачи расстояние будет вычисляться следующим образом:

$$\forall x, y \in cities \quad dist(x, y) = ||x - y||_2 = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Итоговая формулировка задачи:

Евклидово TSP
Дано: N точек в $\mathbb{R}^2$ с евклидовым расстоянием, т.е. $dist(x, y) = ||x - y||_2$
Найти: наикратчайший гамильтонов цикл

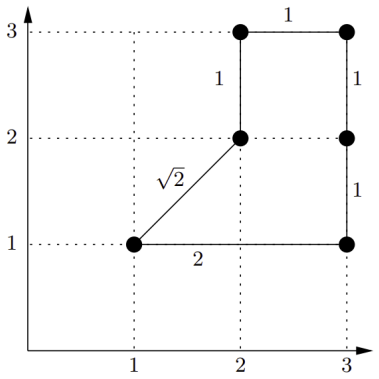

Рис. 1. Пример евклидовой TSP с $D = 6 + \sqrt{2}$

Евклидово TSP является **NP**-трудным, как и общий TSP, так как сводимость к языку HAMCYCLE[2] из **NP** производиться через тот же сертификат – сам цикл.

Сумма радикалов определяется как конечная линейная комбинация радикалов $\sqrt{x_i}$, где $x_i \in \mathbb{R}_+$

Существует открытая проблема: можно ли точно вычислить сумму радикалов за полиномиальное время, так как нет алгоритма эффективно вычисляющего точное значение корня числа.

Из примера евклидовой TSP на рис. 1 мы заключаем, что длина маршрута может быть иррациональной. Поэтому даже когда входные точки имеют целочисленные координаты, расстояния между ними обычно являются иррациональными числами, а длина маршрута представляет собой сумму радикалов, что затрудняет выполнение символьных вычислений, необходимых для точного сравнения длин разных маршрутов. Следовательно, проблема с суммами радикалов является препятствием для доказательства того, что евклидово TSP находится в **NP** и, следовательно, является **NP**-полным. Дискретизированная версия задачи с целочисленными расстояниями является **NP**-полной [2]. Известно, что с рациональными координатами евклидово TSP находится в иерархии подсчета [3], в подклассе **PSPACE**. С произвольными вещественными координатами

[2] HAMCYCLE = $\{G \mid$ в графе G существует гамильтонов цикл$\}$

евклидова TSP не может быть в таких классах, поскольку существует несчетное множество возможных входных данных. Несмотря на эти сложности, аппроксимация ответа для евклидовой TSP намного проще, чем для Δ-TSP с произвольной метрикой [4].

2. Муравьиные алгоритмы

2.1 Понятие муравьиного алгоритма

Имитация самоорганизации муравьиной колонии составляет основу муравьиных алгоритмов (*алгоритм оптимизации подражанием муравьиной колонии*, англ. *Ant Colony Optimization*, ACO). Суть подхода заключается в анализе и использовании модели поведения муравьев, ищущих пути от колонии к источнику питания, и представляет собой мета-эвристическую оптимизацию. Первая версия муравьиного алгоритма, предназначенная для поиска приближенного решения задачи коммивояжера, была разработана доктором наук Марко Дориго в 1992 году [5].

Биологический прототип

Муравьи относятся к социальным насекомым, образующим коллективы. Колония муравьев может рассматриваться как многоагентная система, в которой каждый агент, а именно муравей, функционирует автономно по очень простым правилам. Несмотря на примитивность поведения каждого индивида, поведение всей системы представляет собой высокоструктурированную социальную организацию. Колония муравьев способна успешно выполнить задачу поиска оптимального пути между гнездом и источником пищи, в то время как один муравей, вероятно, потерпит неудачу, особенно учитывая тот факт, что муравьи почти слепы.

Было обнаружено, что при движении муравьи оставляют след из специального вещества – *феромона*. Этот след может быть замечен другими муравьями. Тогда при попадании на феромонный путь во время случайного блуждания муравей с высокой вероятностью продолжит движение по следу. Таким образом, феромон играет роль *положительной обратной связи* в системе – чем больше муравьев движется по помеченному пути, тем привлекательнее он становится для других муравьев. В результате с течением времени большая часть муравьев будет передвигаться от муравейника до найденного источника пищи по одному и тому же пути.

Такой способ коммуникации муравьев в колонии (через феромон) называется стигмергией [6].

Стигмергия (англ. stigmergy) представляет собой механизм спонтанного непрямого взаимодействия между индивидами, заключающийся в оставлении меток в некоторой области окружающей среды, стимулирующих дальнейшую активность других индивидов при попадании в эту область. Таким образом, информация остается локально распределенной.

Стигмергия является одной из форм самоорганизации, создающей сложные структуры, но без какого-либо планирования, контроля, или даже прямой связи между индивидами. Также важно, что за счет испарения феромона

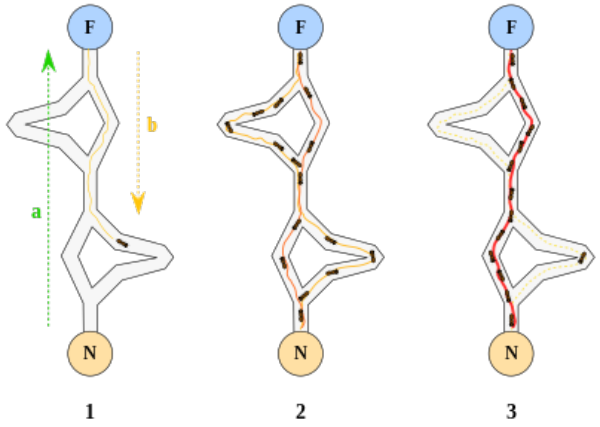

Рис. 2. Схема движения муравьев между гнездом (Nest) и пищей (Food) с отображением феромонного следа

гарантируется, что найденное локально оптимальное решение не будет единственным – муравьи будут искать и другие пути, а не двигаться одним и тем же субоптимальным маршрутом. Это является *отрицательной обратной связью* в системе. Таким образом, самоорганизация является результатом взаимодействия следующих четырех компонентов:

- случайность;
- многократность;
- положительная обратная связь;
- отрицательная обратная связь.

Важной особенностью поведения муравьев является то, что найденный ими путь иногда оказывается наикратчайшим путем, связывающим муравейник с источником пищи.

2.2 Муравьиный подход к задаче коммивояжера

Описанная схема коммуникации муравьев легла в основу муравьиного алгоритма. Чтобы построить этот алгоритм для задачи коммивояжера, берется колония из m муравьев. Многократность взаимодействия реализуется итерационным поиском маршрута одновременно всеми муравьями. При этом каждый муравей рассматривается как независимый индивид, решающий задачу. Поэтому каждый муравей случайным образом помещается в одну из вершин графа, в котором будет искаться минимальный по длине гамильтонов цикл. Все ребра помечаются одинаковым значением феромона τ_0. После этого муравьи начинают перемещаться по ребрам графа. Для каждого муравья переход из вершины i в вершину j зависит от трех составляющих: памяти муравья (*список запретов*), следа феромона и видимости вершин.

• **Список запретов** – это список, в который помещаются номера ранее посещенных муравьем вершин. Используя этот список, муравей гарантированно не попадет в один и тот же город дважды. Ясно, что список запретов муравья возрастает при совершении маршрута и обнуляется в начале каждой итерации алгоритма. Обозначим через $S_{i,k}$ список городов, которые еще необходимо посетить муравью с номером k, находящемуся в вершине i. Очевидно,

что $S_{i,k}$ является дополнением к списку запретов.

• **След феромона** на ребре (i, j) – величина, отражающая подтвержденный опыт муравьев. Она определяет желание муравья, стоящего в вершине i продолжить движение в вершину j. След феромона является глобальной и динамической информацией, то есть она изменяется после каждой итерации алгоритма, отражая приобретенный муравьями опыт. Концентрация феромона на ребре (i, j) на итерации t обозначим через $\tau_{ij}(t)$. Как было сказано ранее, при запуске алгоритма в качестве начального значения феромона для всех ребер берется фиксированное τ_0.

• **Видимость** – величина, обратная расстоянию. То есть для ребра (i, j) видимость $\eta_{ij} = 1/dist(c_i, c_j)$. Видимость является локальной статической информацией, выражающей эвристическое желание муравья посетить город j из города i. Логично, что величина определена именно таким образом, так как в задаче TSP чем ближе город, тем должно быть больше желание его посетить.

Тогда вероятность перемещения k-го муравья в вершину j сразу после посещения вершины i на t-й итерации определяется следующей формулой:

$$p_{ij,k}(t) = \begin{cases} \frac{(\tau_{ij}(t))^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum\limits_{l \in S_{i,k}} (\tau_{il}(t))^{\alpha} \cdot (\eta_{il})^{\beta}} & \text{если } j \in S_{i,k} \\ 0 & \text{иначе} \end{cases} \tag{2}$$

где α, β – это два регулируемых параметра, задающие веса следа феромона и видимости при выборе маршрута. Если $\alpha = 0$, то всегда будет выбираться ближайший город, что соответствует жадному алгоритму. Если $\beta = 0$, то на вероятность влияет лишь феромонное усиление, которое при $\alpha > 1$ влечет за собой быстрое вырождение маршрутов к одному субоптимальному решению [7].

Обратим внимание, что правило (2) определяет лишь вероятности выбора того или иного города. Собственно перемещение осуществляется рандомизированно в соответствии с вероятностями выбора каждого из городов. Заметим, что хотя правило (2) не изменяется на протяжении всей итерации, значения вероятностей $p_{ij,\cdot}(t)$ для двух муравьев в одном и том же городе могут отличаться, так как $p_{ij,k}(t)$ – функция от списка еще не посещенных городов определенного k-го муравья.

После завершения движения каждый муравей откладывает на ребре феромон. Тогда изменение концентрации феромона на ребре определяется следующим образом:

$$\Delta\tau_{ij,k}(t) = \begin{cases} \frac{Q}{D_k(t)} & \text{если } (i, j) \in P_k(t) \\ 0 & \text{иначе} \end{cases} \tag{3}$$

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

Для обеспечения отрицательной обратной связи необходимо учесть испарение феромона – уменьшение во времени количества отложенного на предыдущих итерациях феромона. Обозначим коэффициент испарения феромона через $\rho \in [0, 1]$. Тогда итоговое обновление феромона на ребре (i, j) после того, как все m муравьев закончили свои маршруты принимает вид:

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij,k}(t) \tag{4}$$

Перемещение муравьев по графу и обновления феромона представляют собой одну итерацию муравьиного алгоритма. Важно отметить, что общее количество муравьев в колонии остается постоянным на протяжении выполнения алгоритма. Итерации повторяются до тех пор, пока не окажется выполненным какой-нибудь из критериев останова алгоритма – исчерпано число итераций, достигнута нужная точность, время на работу алгоритма истекло.

По сравнению с точными методами, например динамическим программированием или методом ветвей и границ, муравьиный алгоритм находит близкие к оптимуму решения за значительно меньшее время даже для задач небольшой размерности ($N < 20$). Время оптимизации муравьиным алгоритмом является полиномиальной функцией от размерности $O(t \cdot N^2 \cdot m)$, тогда как для точных методов зависимость экспоненциальная.

Реализация алгоритма

Ниже приводится псевдокод муравьиного алгоритма для задачи TSP, реализующий все идеи описанные ранее. Важно отметить, что рассмотренный муравьиный алгоритм является классической версией муравьиного алгоритма и известен как *муравьиная система* (англ. Ant System, AS).

```
<<Getting info about the TSP task:
  n_cities, dist(i, j)>>

<<Initializing parameters: alpha, beta,
  rho, Q, tau_0, n_ants, n_iters>>

<<Set the initial pheromone, visibility>>

best_dist = 'inf'

# main loop
for i in range(n_iters):
    best_iter_dist = 'inf'
    ants_position = randint.rvs(0, n_cities,
    size=n_ants)

    # launching ants
    for ant in range(n_ants):
        ant_pos = ants_position[ant]
        <<Build a route T_ant(i) according to
          rule (2), calculate its length
          D_ant(i)>>
        best_iter_dist = min(D_ant(i),
    best_iter_dist)

    # updating the distance if a better one was found
    best_dist = min(best_iter_dist, best_dist)
    <<Update pheromone traces for all edges
      according to rule (3)>>

# returning result
print("Path length:", best_dist)
```

Муравьиный алгоритм был реализован на языке Python. Исходный код расположен в аннотации статьи. Также была реализована генерация тестов, где по заданному количеству N вершин и границам прямоугольника генерируются координаты вершин графа из равномерного распределения внутри прямоугольника.

2.3 Модификации муравьиного алгоритма

Рассмотренная ранее модель муравьиных вычислений не является единственной. С момента ее создания были разработаны и другие модели, основанные на той же принципиальной идее.

Элитная муравьиная система

Качество решений, производимых муравьиной системой, можно улучшить, используя некоторое количество элитных муравьев. Такая модификация называется элитной муравьиной системой (Elitist Ant System, EAS, [5]). Идея элитарной стратегии в контексте муравьиной системы состоит в том, чтобы уделять дополнительное внимание наилучшему пути, найденному после каждой итерации. Поскольку вполне вероятно, что некоторые ребра этого пути являются частью оптимального решения. Таким образом, когда происходит обновление феромона (по формуле 4), наилучший путь обрабатывается так, как если бы определенное количество муравьев, а именно элитных муравьев, выбрали этот путь. Цель состоит в том, чтобы направлять поиск в последующих итерациях.

Пусть e – число элитных муравьев, $P^*(t)$ – наилучший маршрут с начала работы алгоритма до t-ой итерации включительно, а $D^*(t)$ – длина этого наилучшего маршрута. Тогда обновление феромона имеет следующий вид:

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{k=1}^{m}\Delta\tau_{ij,k}(t) + \Delta\tau^*_{ij}(t)$$

где

$$\Delta\tau^*_{ij}(t) = \begin{cases} \frac{Q+e}{D^*(t)} & \text{если } (i,j)\in P^*(t) \\ 0 & \text{иначе} \end{cases}$$

Результаты вычислений, представленные Dorigo (1996) [7], выявили, что использование стратегии элитных муравьев позволяет как находить лучшие маршруты, так и получать их за меньшее число итераций.

Ранговая муравьиная система

Ранговая модификация классического муравьиного алгоритма (Rankbased Ant System, AS_{rank}, [8]), состоит в том, что в конце каждой итерации муравьи ранжируются в соответствие с длинами пройденных ими путей. Логично, что лучшие муравьи после ранжирования – это муравьи, прошедшие по более коротким путям. Тогда количество феромонов, оставляемое муравьем на ребрах, назначается пропорционально его позиции в рейтинге ранжирования. Кроме того, для более тщательного исследования окрестностей уже найденных удачных решений, алгоритм использует элитных муравьев. Количество элитных муравьев определяет то, сколько лучших после ранжирования муравьев будут выделять феромон. Таким образом, можно избежать чрезмерно выделенных феромонами ребер, вызванных тем, что многие муравьи используют субоптимальные пути.

Пусть e – число элитных муравьев, r – ранг муравья, $P^r(t)$ – маршрут, найденные муравьем с рангом r, $D^r(t)$ – длина этого маршрута, а $\Delta\tau^*_{ij}(t)$ – определен так же как и в EAS. В такой комбинированной постановке, с элитизмом и ранжированием, новое обновление феромонов выполняется в соответствии со следующей формулой:

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{r=1}^{e}\Delta\tau^r_{ij}(t) + \Delta\tau^*_{ij}(t)$$

где

$$\Delta\tau^r_{ij}(t) = \begin{cases} (e-r)\frac{Q}{D^r(t)} & \text{если } (i,j)\in P^r(t) \\ 0 & \text{иначе} \end{cases}$$

Представленная процедура обновления является хорошим компромиссом. Результаты исследования, представленного в [8], показывают, что "эксплуатация" (за счет выделения хороших путей), а также "разведка" (за счет расширения акцента на нескольких хороших муравьях) значительно высоки и хорошо сбалансированы.

2.4 Исследование параметров Ant System

Как уже было замечено, в муравьиной системе и всех ее модификациях присутствуют различные параметры, участвующие в формулах пересчета феромона или накладывающие некоторые ограничения на разные этапы алгоритма. В частности, речь идет о параметрах: $m, \tau_0, \alpha, \beta, \rho, Q, e$. Выбор правильного соотношения параметров является предметом исследований, и в общем случае производится на основании экспериментов.

Экспериментальное исследование различных алгоритмов ACO для TSP, приведенное в книге [9], выявило настройки параметров, которые зачастую приводят к хорошей производительности. Для параметров, которые являются общими почти для всех алгоритмов ACO, хорошие настройки (если локальный поиск не применяется) приведены в следующей таблице.

ACO algorithm	α	β	ρ	m
AS	1	[2, 5]	0.5	N
EAS	1	[2, 5]	0.5	N
AS_{rank}	1	[2, 5]	0.1	N

Таблица 1. Стандартная настройка параметров для ACO

Здесь N – это число городов в задаче коммивояжера. Все варианты AS также требуют некоторых дополнительных параметров. Хорошими значениями для этих параметров являются:

- EAS: количество элитных муравьев $e = N$
- AS_{rank}: количество муравьев, выделяющих феромоны, составляет 6

Несмотря на то, что эти параметры обеспечивают приемлемую производительность в значительном наборе экземпляров TSP, важно понимать, что в отдельных случаях

различные настройки могут привести к гораздо большей производительности. Например, рассмотрим запуск AS на задаче коммивояжера из 16 вершин. По рис. 3 становится понятно, что заявленное значение $\alpha = 1$ приводит к куда более плохому результату нежели $\alpha = 0.1$. Более того, при $\alpha = 1$ муравьиная система даже не смогла никак оптимизировать изначальное приближение. Это можно объяснить тем, что вклад феромона в выборе ребра для перехода оказался очень большим, а для сравнительно небольшой задачи в 16 городов (см. рис. 4), где все вершины равномерно дистанцированы друг от друга, намного важнее видимость вершин.

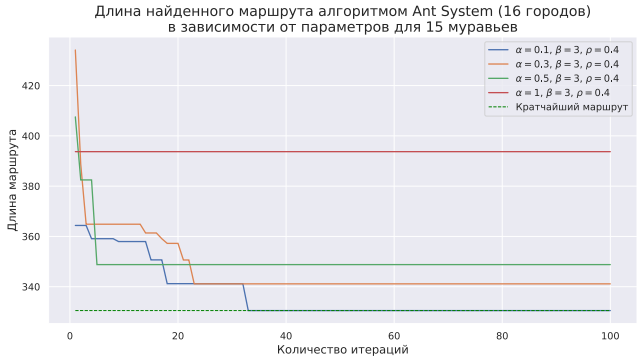

Рис. 3. Решения AS для различных α

Рис. 4. Найденный оптимальный маршрут для 16 городов

Также на основании тестов на рассматриваемых задачах ($N < 150$) было выявлено, что более хорошие показатели достигаются при количестве муравьев немного меньшем, чем количество городов (см. рис. 5). Однако важно понимать, что чем больше агентов участвуют в поиске решения, тем вероятнее получить более точную аппроксимацию ответа. Но когда речь идет об оптимальном времени работы алгоритма, а как было отмечено ранее, оно напрямую зависит от количества муравьев, то необходимо находить компромиссный подход.

Далее важно продемонстрировать, что AS не обладает сходимостью решения, что подтверждает факт именно мета-эвристического подхода решения TSP. Проведенные эксперименты на коммивояжере из 16 городов свидетельствуют, что популяция решений никогда не вырождается к одному, общему для всех муравьев маршруту. Наоборот, алгоритм продолжает синтезировать новые, возможно лучшие решения. На рис. 6 синей линией показаны наилучшие решения, найденные на каждой итерации алгоритма AS с

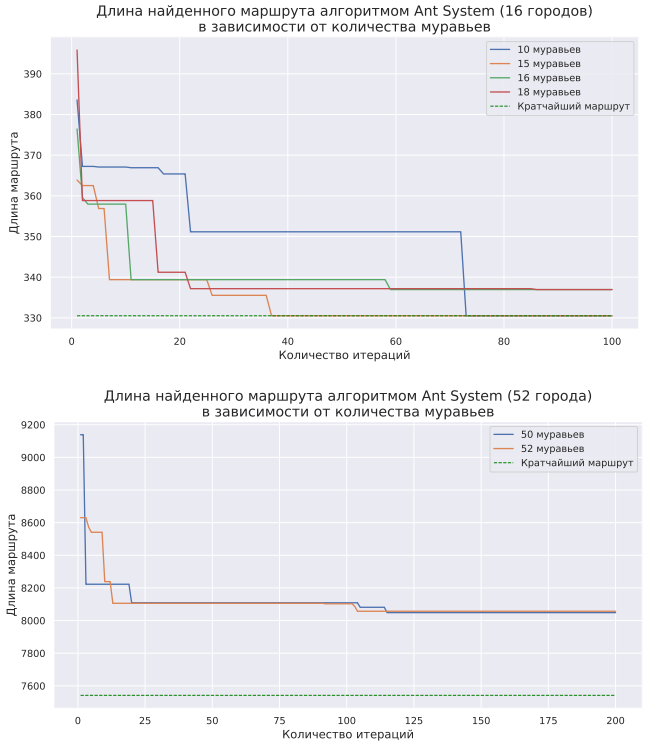

Рис. 5. Сравнение количества муравьев в AS

параметрами $\alpha = 0.1, \beta = 3, \rho = 0.4, m = 15$. Красной линией показаны наилучшие решения, найденные с начала работы алгоритма. Как видно, линии не совпадают, следовательно, муравьиный алгоритм генерирует новые решения на каждой итерации. Об этом свидетельствует и рис. 7, на котором показано среднеквадратическое отклонение длин маршрутов, найденных муравьями на текущей итерации алгоритма. Даже если оптимальный маршрут уже найден, алгоритм продолжает поддерживать разнообразность популяции решений.

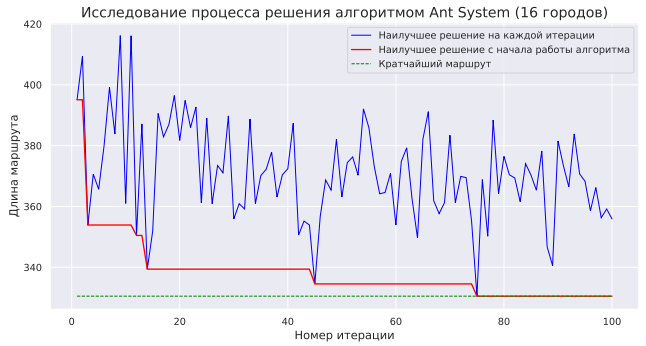

Рис. 6. Динамика нахождения решения AS

Для дальнейшего более качественного исследования алгоритмов воспользуемся тестами, представленными на сайте [10].

Для начала продемонстрируем работу муравьиной системы на тесте ”Lin-105” [10]. Из рис. 8 можно заклю-

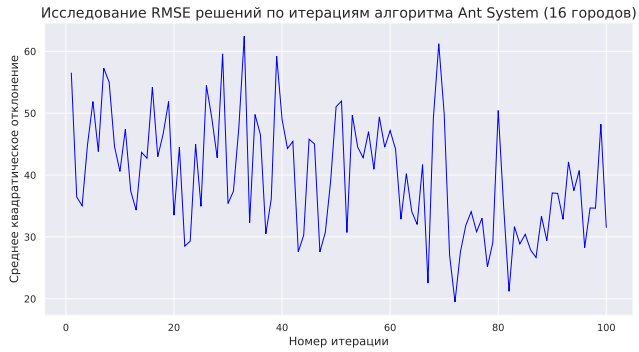

Рис. 7. Разброс решений AS

чить, что алгоритм работоспособен и на больших задачах. Единственная трудность, которая возникает, это корректный подбор параметров, для получения результата наиболее приближенного к оптимальному. Тем не менее AS (с параметрами $\alpha = 0.5$, $\beta = 5$, $\rho = 0.2$, $m = 100$) удалось достичь достаточно хорошей оценки 15341 оптимального маршрута длинной 14379.

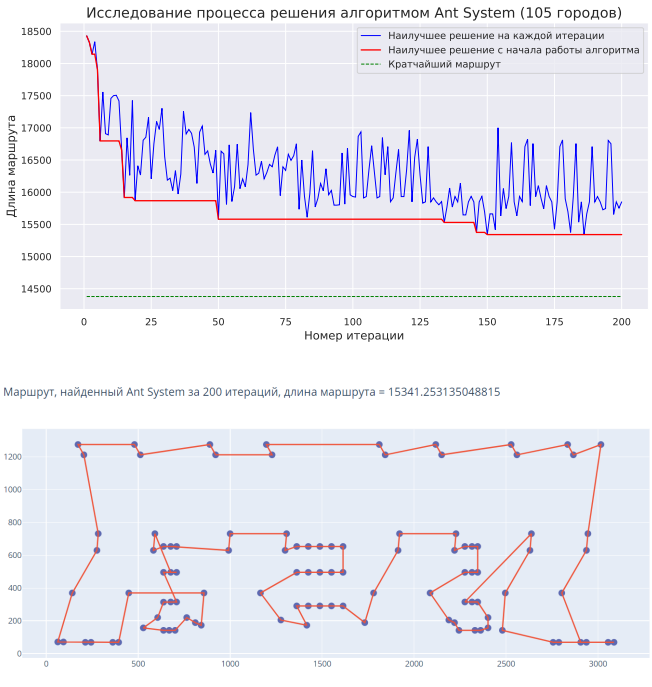

Рис. 8. Решение AS на тесте ”Lin-105” [10]

Чтобы сравнить эффективность различных модификаций муравьиного алгоритма (AS, EAS и AS_{rank}) был произведен подбор параметров (α, β, ρ) по сетке, для каждого алгоритма по отдельности на тесте ”Berlin-52” [10]. Было принято решение работать на коммивояжере меньшей размерности, потому что на 105 городах подбор параметров только для AS занял более 4 часов.

Для начала имеет смысл рассмотреть EAS, чтобы продемонстрировать актуальность ограничений из предыдущего раздела. Из рис.9 видно, что добавление лишь пары элитных муравьев ухудшает поиск аппроксимации. Поэтому замечание про то, что в общем случае количество элитных муравьев должно совпадать с количеством всех агентов в системе вполне справедливо.

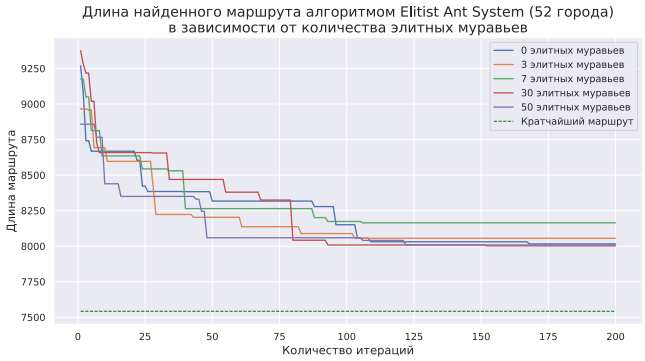

Рис. 9. Сравнение количества элитных муравьев на тесте ”Berlin-52”

Далее для AS_{rank} на тесте ”Berlin-52” [10] был произведен подбор количества элитных муравьев по сетке. Наилучшую эффективность по результатам 5 запусков для каждого фиксированного e показала AS_{rank} с $e = 7$ (таблица 2). Таким образом, мы снова подтверждаем эксперименты в книге [9] о количестве элитных муравьев для ранговой муравьиной системе.

Elitist	3	5	6	7	9
Маршрут	8138.4	8082.1	8014.5	**7921.8**	8002.0

Таблица 2. Количество элитных муравьев для AS_{rank} на 100 итерациях (истинный ответ: 7542)

Представим на одном графике сравнение всех трех модификаций с лучшими параметрами для каждой версии муравьиного алгоритма.

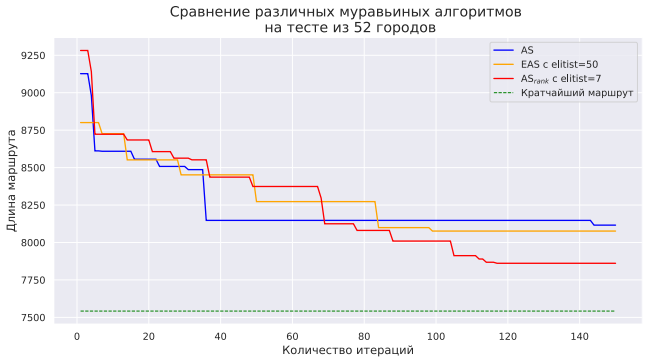

Рис. 10. Сравнение модификаций на тесте ”Berlin-52” [10]

Как видно из рис. 10 ранговая муравьиная система нашла наиболее оптимальное решение. Логично заключить, что это является следствием хороших модификаций базового алгоритма на основе использования подхода элитных муравьев и хорошего подбора параметров для конкретного теста ”Berlin-52” [10].

3. Генетические алгоритмы

3.1 Понятие генетического алгоритма

Генетический алгоритм является мета-эвристическим алгоритмом, основанным на принципах эволюции и естественного отбора. Несмотря на свою рандомизированность, он использует многие природные закономерности, что позволяет ему приходить к субоптимальному решению. Генетический алгоритм был впервые разработан Джоном Холландом, его коллегами и студентами в Мичиганском университете. Их целью было построить абстракцию, отражающую процессы адаптации в природе и создать искусственную систему, следующую этим закономерностям.

В настоящее время генетический алгоритм находит все больше применений: в бизнесе, науке, инженерии и т.д. Это объясняется не только его «природной красотой», но и балансом между точностью и эффективностью, а так же отсутствием ограничений на пространства поиска решений (например, он не требует существование производных в отличие от многих задач оптимизации). [11]

Биологический прототип

Как уже упоминалось выше, генетический алгоритм симулирует существование некоторой популяции особей, которые мутируют, размножаются и погибают. Алгоритм можно разделить на 5 основных этапов

• **Инициализация популяции**: Начальная популяция обычно выбирается случайно, а ее размер подбирается исследователем в зависимости от задачи (обычно это число порядка сотни или тысячи). Если необходимо, можно модифицировать случайный выбор и генерировать больше особей в той области, где более вероятно находится оптимальное решение. Данный этап выполняется один раз, затем начинается цикл, количество итераций которого так же подбирается разработчиком.

• **Отбор (selection)**: Этот этап отвечает за отбор более выносливых особей для размножения.

Функция приспособленности (англ. fitness function) – функция, показывающая насколько оптимально решение в условиях данной задачи

При помощи функции приспособленности f определяется оптимальность решения для каждой особи и с помощью некоторого алгоритма принимается решение, какие из них будут допущены к размножению, а какие – убиты (количество выживших особей задается как гиперпараметр алгоритма). Подробнее о некоторых таких методах будет рассказано дальше в статье

• **Размножение (crossover)**: Теперь когда алгоритм выбрал наиболее выносливых особей необходимо получить потомков, которые унаследуют какие-то черты от родителей и смогут продвинуться ближе к оптимальному решению. Перед алгоритмом стоит две задачи: выбрать родителей для размножения и сгенерировать ребенка, который унаследует характеристики обоих своих предков.

Для выбора родителей есть несколько универсальных приемов: панмиксия (оба выбираются случайно), инбридинг (первый родитель выбирается случайно, а вторым выбирается наиболее похожий на него) и аутбридинг (второй родитель выбирается так, чтобы он был наименее похожим на первого). Для создания потомка универсальных методов уже нет: так как каждая задача специфична и особи содержат разную информацию и характеристики, нужно подбирать свой способ размножения для каждой конкретной ситуации.

• **Мутации (mutations)**: Мутации в генетическом алгоритме используются для того, чтобы внести разнообразие в популяцию. Без них алгоритм может найти локальный экстремум, и вся популяция заполнится копиями этой особи. Мутации не допускают этого: выбирается некоторая доля мутантов в популяции (параметр подбирается исследователем) и генотип каждого из них меняется по некоторым правилам, специфическим для каждой задачи.

3.2 Применение алгоритма к задаче коммивояжера

В данном разделе подобнее расписываются конкретные методы для реализации этапов алгоритма описанных ранее и выбор более оптимальных из них для решения метрической задачи коммивояжера.

Особи в описанной реализации будут представлены как некоторая перестановка городов, которая и будет задавать путь коммивояжера.

Реализации отбора (селекции)

Рассмотрим 3 универсальных метода селекции в генетических алгоритмах • **Турнирная селекция:** случайно выбирается некоторое количество особей (обычно две) и из них выживает сильнейшая

• **Метод рулетки:** выбирается нужное количество особей из популяции с возвращением, вероятность выбора каждой особи равна

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{5}$$

• **Метод ранжирования:** вероятность выбора особи зависит от ее позиции в списке, который отсортирован по значению f

$$p_i = \frac{1}{N}\left(a - (a-b)\frac{i-1}{N-1}\right), \tag{6}$$

где $a \in [1,2], b = 2 - a, i$ – порядковый номер особи в отсортированном списке (формула дана для случая минимизации функции приспособленности)

Реализации размножения (crossover)

Вернемся к одному из самых интересных этапов алгоритма – размножению. Как уже говорилось ранее, этот этап зависит от конкретной задачи. Для задачи коммивояжера уже придумано очень много разных способов скрещивания особей, в статье будут описаны самые эффективные из них.

Для начала поймем, какую характеристику мы хотим передать наследнику. Основные операторы скрещивания для задачи коммивояжеры делятся на несколько типов:

1. сохраняющие абсолютную позицию городов в пути
2. сохраняющие относительный порядок
3. сохраняющие ребра

Абсолютную позицию в задаче коммивояжера сохранять нелогично (этот тип просто перенесся из достаточно универсальных типов скрещивания для генетических алгоритмов) и неудобно (часто в дальнейшем приходится избавляться от дубликатов городов в пути). Рассмотрим представителей двух остальных типов: Order crossover и Edge recombination crossover

• **Order crossover (OX).** В данном методе выбирается случайная подстрока в пути первого родителя, а остальные позиции заполняются неиспользованными городами в том порядке, в каком они встречаются во втором родителе начиная с позиции конца подстроки.

Рассмотрим простой пример для понимания: скрестим пути 12564387 и 14236578. Пусть алгоритм выбрал подстроку 564 в первом родителе. Тогда путь в потомке будет иметь вид 23564781

• **Edge recombination crossover (ER).** Данный тип скрещивания пытается сохранить как можно больше ребер, которые есть хотя бы в одном из родителей.

Если брать такие ребра случайно, они могут быстро закончится, поэтому нужно реализовать алгоритм так, чтобы каждое взятое ребро не слишком сильно сокращало количество потенциальных вариантов. Для этого вводится некоторая структура данных позволяющая последовательно выбирать ребра так, чтобы они отнимали как можно меньше возможных вариантов (при одинаковом количестве выбор делается случайно). Более подробно об устройстве этой структуры можно прочитать в статье [12] или посмотреть в реализации, приведенной в аннотации к данной статье.

Реализация мутаций

Рассмотрим 3 основных оператора мутации [12]

• **Swap.** Самый простой и очевидный метод, в котором просто меняются местами две случайных вершины из пути

• **Scramble.** Случайно выбираются 2 вершины, подстрока пути между которыми перемешивается

• **Local hill-climbing (2-opt).** Этот подход опирается на идею алгоритма 2-opt, который используется для решения задачи коммивояжера. Его идея заключается в итеративном выборе пары ребер и перенаправлении их так, как показано на рис. 11, чтобы маршрут остался валидным, и его длина уменьшилась.

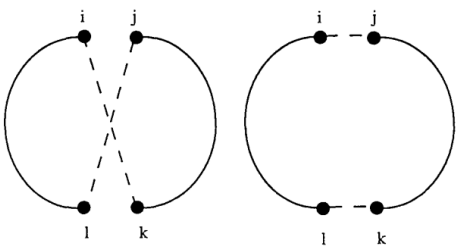

Рис. 11. Замена ребер $(i,k),(j,l)$ на $(i,j),(k,l)$

Данная мутация применяет одну (или несколько) итераций этого алгоритма к пути

3.3 Реализация алгоритма

Ниже приводится псевдокод генетического алгоритма для задачи TSP, реализующий идеи описанные ранее.

```
<<Getting info about the TSP task:
  n_cities, dist(i, j)>>

<<Initializing parameters: survived, mutated,
population_size, stages of algorithm>>

<<generate initial population>>

best_dist = 'inf'

# main loop
for i in range(n_iters):
    <<select survived * population_size fittest
    species>>

    <<generate enough pairs for crossover>>
    pairs = <<enough to restore population>>
    for first, second in parent_generator(
    population, pairs):
        <<create child using crossover operator
    >>

    <<make mutations in mutated *
    population_size fittest species>>


    # updating the distance if a better one was found
    best_dist = min(best_iter_dist, best_dist)

# returning result
print("Path length:", best_dist)
```

Алгоритм был реализован на языке Python. Исходный код расположен в аннотации статьи.

3.4 Подбор параметров и оптимизация

Основные параметры генетического алгоритма, которые можно подбирать - это реализации каждого из 5 этапов (у некоторых реализаций есть свои гиперпараметры, подбором которых тоже можно заниматься), размер популяции, количество итераций работы, доля отобранных для размножения особей, а также доля мутантов.

Для выбора наилучших реализаций этапов алгоритма все различные комбинации были запущены на небольшом тесте (из 16 вершин) несколько раз. Как и ожидалось [12], наилучшей оказалась следующая комбинация: метод рулетки для селекции, метод инбридинга для выбора родителей, edge recombination скрещивание и 2-opt мутация (см. рис. 12).

Уже с использованием данных этапов был запущен перебор других гиперпараметров: доля выживающих особей после селекции и доля мутантов. Оптимальными значениями оказались 0.7 и 0.1 (часть результатов см. в таблице 3, все результаты перебора параметров можно посмотреть в репозитории, который находится в аннотации к статье). Алгоритм с такими параметрами смог найти оптимум на тесте из 16 вершин.

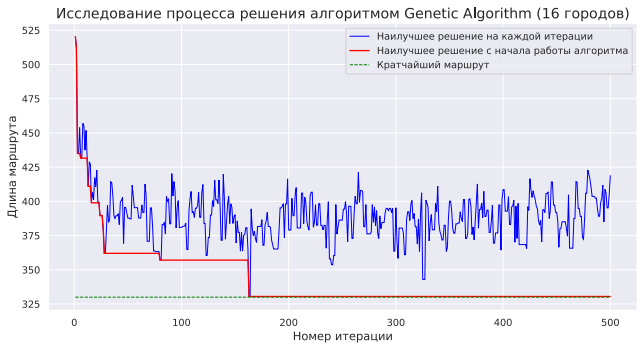

Рис. 12. Работа оптимальной реализации алгоритма со случайно инициализацией популяции

Survived	Mutated	Mean	Min	Max
0.7	0.1	330.5	330.5	330.5
0.5	0.3	332.3	330.5	334.5
0.7	0.7	385.2	376.2	397.1

Таблица 3. Подбор гиперпараметров генетического алгоритма с оптимальными этапами на тесте из 16 вершин (истинный ответ: 330.5)

При переходе к тестированию на больших данных (50 и более вершин) обнаружилось, что алгоритм очень далек от оптимального решения. Частично, это объясняется тем, что начальная популяция генерировалась случайно и соответственно изначальные пути были очень неэффективные (см. рис. 13) и имели не очень много ценной информации в генах, вследствие чего, алгоритм искал некоторый локальный оптимум среди этих неоптимальных путей.

Для оптимизации была применена новая идея: использование жадного алгоритма для генерации популяции (путь генерируется выбором на каждом шаге ближайшей вершины) [13]. Таким образом, в изначальной популяции уже будет некоторое достаточно хорошее решение, которое алгоритм будет пытаться оптимизировать.

Для этого способа инициализации был запущен новый подбор параметров на тесте ”Berlin-52” [10], некоторые результаты можно увидеть в таблице 4 (в целом было запущено 54 вариации алгоритма со всеми возможными комбинациями этапов)

Selection	Cross.	Mut.	Mean	Min	Max
Roulette	In/OX	2-opt	7856.9	7785	7902
Rank	Pan/OX	2-opt	8137.2	8079.1	8182.2
Tournament	In/ER	Swap	8182.2	8182.2	8182.2

Таблица 4. Подбор этапов генетического алгоритма на тесте ”Berlin-52” [10] (лучший ответ жадного алгоритма: 8182.2, истинный ответ: 7542)

Можно заметить, что жадный алгоритм уже дает достаточно хорошее приближение, и некоторые модификации алгоритма не могут оптимизировать его (в некоторых ситуациях мутации и скрещивания могут только ухудшать

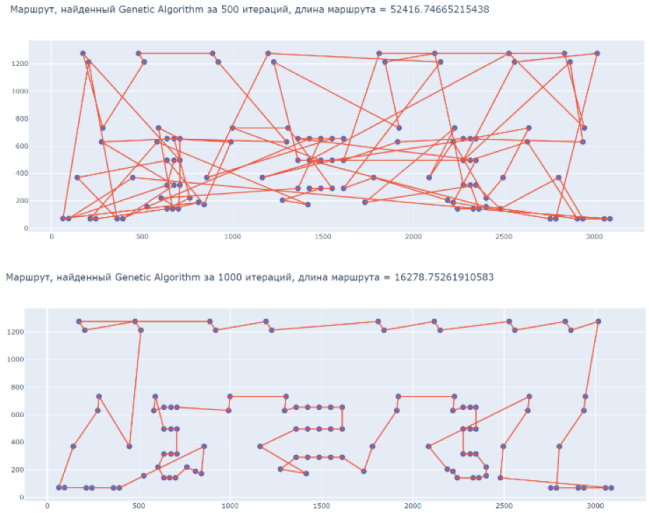

Рис. 13. Результат работы алгоритма со случайным созданием популяции и с использованием жадного алгоритма на тесте ”Lin-105” [10]

алгоритм (см. рис. 14)). Для самого эффективного алгоритма был снова запущен перебор остальных гиперпараметров и были выбраны оптимальные (см. таблицу 5).

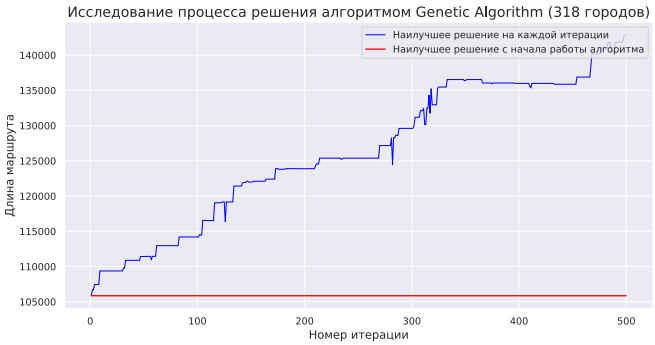

Рис. 14. Неэффективная модификация алгоритма, которая не смогла оптимизировать жадное решение

Так же хотелось бы отметить, что почти во всех проведенных тестах среди алгоритмов селекции лучший результат показывал метод рулетки (что подтверждает другие исследования по этой теме [14]), а среди мутаций - 2-opt мутация (это объясняется тем, что остальные операторы, мутации представленные в статье, не так сильно опираются на внутреннюю структуру задачи коммивояжера).

На рис. 15 приведен график работы алгоритма с использованием жадной инициализации популяции с оптимальными параметрами. Его вид очень сильно отличается от графиков, показанных ранее: здесь отчетливо видны ступеньки, отражающие этапы поиска решения.

Из-за того, что в исходной популяции не очень много разнообразия видим, что часто получается одно и то же оптимальное решение, и когда создается достаточное количество копий этой особи, она сохраняется в популяции.

Survived	Mutated	Mean	Min	Max
0.3	0.1	7842.4	7785	7883.6
0.7	0.1	7913	7883	7972.4
0.5	0.3	8094.4	8076.1	8156.9

Таблица 5. Подбор гиперпараметров генетического алгоритма с оптимальными этапами на тесте ”Berlin-52” [10] (лучший ответ жадного алгоритма: 8182.2, истинный ответ: 7542)

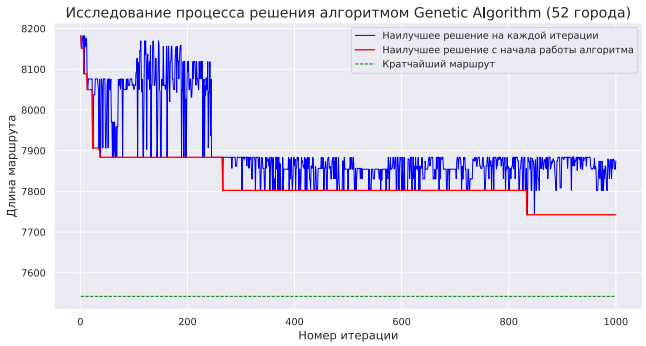

Рис. 15. Работа алгоритма с использованием жадного создания исходной популяции

4. Сравнение муравьиного и генетического алгоритмов

Для анализа эффективности муравьиных и генетических алгоритмов на задачах о коммивояжере были проведены запуски каждого из алгоритмов на оптимальных параметрах (полученные при исследовании алгоритмов) с расчетом времени работы на каждой итерации. Далее результаты были нанесены на одну сетку по времени.

Рассмотрим задачу коммивояжера на 16 вершинах.

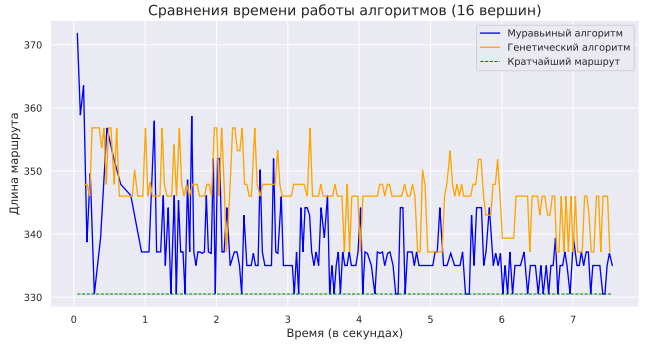

Рис. 16. Сравнение времени работы ACO и GA на тесте из 16 вершин

Из рис. 16 можно заключить, что муравьиный алгоритм на маленьких задачах быстрее решает поставленную задачу, так как менее чем за секунду был найден оптимальный маршрут . У генетического алгоритма видна тенденция убывания, однако минимум за рассматриваемое время не был достигнут.

Рассмотрим задачу коммивояжера ”Berlin-52” [10].

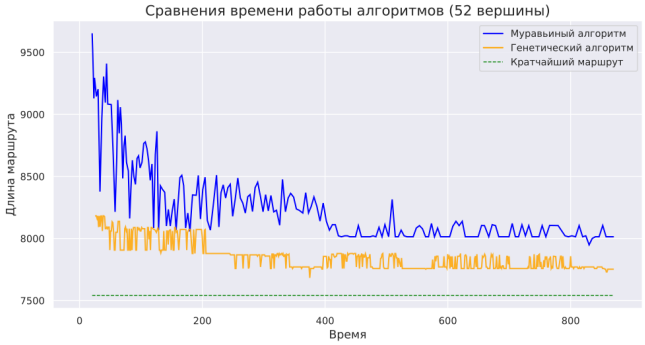

Рис. 17. Сравнение времени работы ACO и GA на тесте ”Berlin-52” [10]

На рис. 17 видно, что генетический алгоритм справился лучше. Это частично объясняется тем, что подбор параметров производился именно на этом тесте. Но все же видно, что так как самая эффективная версия генетического алгоритма, описанная в данной статье использует жадный алгоритм как начальное решение для оптимизации, он быстрее справляется с оптимизацией, чем муравьиный алгоритм.

Рассмотрим задачу коммивояжера ”Lin-105”. [10].

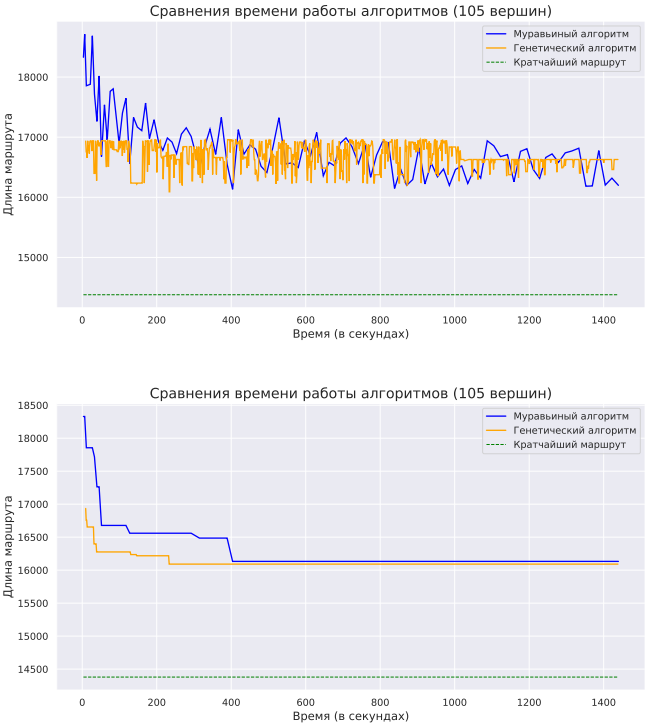

Рис. 18. Сравнение времени работы ACO и GA на тесте ”Lin-105”. [10]. На первом графике длина лучшего маршрута на итерации. На втором графике длина лучшего маршрута с начала работы алгоритма

Из рис. 18 можно заключить, что алгоритмы справились примерно одинаково. Однако анализируя форму графиков,

можно заметить, что график работы генетического алгоритма колеблется на одном месте очень долго, в то время как на графике муравьиного видна тенденция к убыванию. С учетом того, что данный тест был запущен на маленьком количестве итераций, можно сказать, что у муравьиного алгоритма больше шансов приблизиться к оптимальному ответу, чем у генетического алгоритма.

5. Результаты

Из проведенных исследований каждого из алгоритмов по отдельности и итогово сравнения эффективности обоих алгоритмов можно сделать следующие выводы:

1. Муравьиный подход к решению задачи коммивояжера является хорошим способом качественно аппроксимировать ответ. Однако на практике близость ответа к оптимальному значению сильно зависит от подобранных параметров, которые приходится для каждой конкретной задачи находить исходя из экспериментов. Более того, существуют различные модификации базового муравьиного алгоритма, вносящие новые параметры в процесс пересчета феромона, от которых тоже зависит эффективность алгоритма.

2. Генетический алгоритм, это очень интересный инструмент для решения всевозможных задач, однако пользоваться им достаточно сложно, так как у него очень много различных параметров, оптимальные значения которых зависят не только от типа задачи, но и от размера и вида входных данных. Так же каждый этап алгоритма можно реализовать различными способами и от выбора реализации сильно зависит эффективность алгоритма. Таким образом, решение задачи с помощью генетического алгоритма представляет собой исследование и подбор наилучших эвристик и параметров для данной ситуации.

3. В некоторых случаях, с точно подобранными параметрами под конкретную задачу генетический алгоритм может достичь большей точности, чем муравьиной, или как минимум сделать это за меньшее время. Однако муравьиный алгоритм изначально был разработан для решения задачи коммивояжера, а генетический алгоритм направлен на решение огромного класса задач, поэтому в среднем муравьиный алгоритм будет работать лучше.

По результатам проделанной работы, можно сформировать **рекомендацию по выбору алгоритма** для расчета длины маршрута в задаче коммивояжера. Критерий будет зависеть от количества городов N и доступного ресурса машинного времени:

- При $N \leq 30$ поиск маршрута целесообразно осуществлять с помощью муравьиного алгоритма
- При $30 < N < 100$ стоит сделать выбор в пользу генетического алгоритма
- При $N \geq 100$:
  - если необходимо точное решение и время работы алгоритма не имеет значения, то лучше использовать муравьиный алгоритм
  - если достаточно получить решение не очень высокой точности и есть ограничения по времени, то разумнее использовать генетически алгоритм

Список литературы

[1] Johnson D. S. and McGeoch L. A. The traveling salesman problem: A case study in local optimization. *Local Search in Combinatorial Optimization*, pages 215–310, 1997.

[2] Christos H. Papadimitriou. The euclidean traveling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[3] Kjeldgaard-Pedersen J. Allender E., Bürgisser P. and Mitersen P. B. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2007.

[4] Larson R. C.; Odoni A. R. 6.4.7: Applications of network models § Routing problems §§ Euclidean tsp. *Urban Operations Research, Prentice-Hall*, 1981.

[5] Dorigo M. Optimization, learning and natural algorithms (in italian). *Ph.D. dissertation, Dipartimento di Elettronica, Politecnico di Milano, Italy*, 1992.

[6] Штовба С. Д. Муравьиные алгоритмы. *Exponenta Pro. Математика в приложениях*, 4:70–75, 2003.

[7] Dorigo M.; Maniezzo V.; Colorni A. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–41, 1996.

[8] Hartl R. F. Bullnheimer B. and Strauss C. A new rank based version of the ant system – a computational study. *Central European Journal of Operations Research*, 7(1):25–38, 1999.

[9] Thomas Stutzle Marco Dorigo. *Ant colony optimization*. A Bradford Book, 2004.

[10] http://comopt.ifi.uni heidelberg.de/software/TSPLIB95/.

[11] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Publishing Company, Inc., 1989.

[12] Jean-Yves Potvin. *Genetic algorithms for the traveling salesman problem*. J.C. Baltzer AG, Science Publishers, 1996.

[13] Ahmad B. Hassanat; Surya Prasath; Mohammed Ali Abbadi; Salam Amer Abu-Qdari and Hossam Faris. An improved genetic algorithm with a new initialization mechanism based on regression techniques. *Information*, 9(7):167, 2018.

[14] Ryan Champlin. Selection methods of genetic algorithms. *Student Scholarship - Computer Science*, 8, 2018.