САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ – ПРОЦЕССОВ УПРАВЛЕНИЯ

КАФЕДРА МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ ЭКОНОМИЧЕСКИХ СИСТЕМ

Корнилов Вадим Юрьевич

Выпускная квалификационная работа бакалавра

Модификация муравьиного алгоритма для динамической задачи коммивояжёра

Направление 010400

Прикладная математика и информатика

Заведующий кафедрой,
доктор физ.-мат. наук,
профессор — Захаров В.В.

Заведующий кафедрой,
доктор физ.-мат. наук,
профессор — Захаров В.В.

Рецензент,
ведущий инженер-
программист ООО
"Интермедиа" — Щегряев А. Н.

Санкт-Петербург

2016

Оглавление

1. ВВЕДЕНИЕ.....3

1.1. История задачи TSP..... 4

1.2. Постановка задачи TSP..... 5

Математическая модель задачи TSP.....6

1.3. Постановка задачи TDTSP.....7

Математическая модель задачи TDTSP.....7

2. ОБЗОР АЛГОРИТМОВ..... 9

2.1. Biology-inspired algorithms..... 9

2.2. Ant Colony Optimization (ACO)..... 11

2.3. Particle Swarm Optimization (PSO)..... 13

2.4. Artificial Bee Colony.....14

2.5. Выводы..... 15

3. МУРАВЬИНЫЙ АЛГОРИТМ ДЛЯ ЗАДАЧИ TDTSP.....17

3.1. Параметры задачи TDTSP..... 17

3.2. Время пути маршрута..... 17

3.3. Жадный алгоритм..... 18

3.4. Алгоритм локального поиска.....19

3.5. Инициализация феромонов.....19

3.6. Основная процедура.....20

4. ДИНАМИЧЕСКАЯ УСТОЙЧИВОСТЬ.....22

4.1. Динамическая устойчивость для эвристического алгоритма..... 23

4.2. Проверка уровня динамической устойчивости.....24

4.3. Результаты проверки на динамическую устойчивость.....25

5. ДИНАМИЧЕСКАЯ АДАПТАЦИЯ..... 27

5.1. Динамическая адаптация для ACO.....27

5.2. Результаты применения динамической адаптации.....28
5.3. Анализ результатов..... 29
6. ЗАКЛЮЧЕНИЕ..... 30
7. СПИСОК ЛИТЕРАТУРЫ..... 31

1. Введение

Объем рынка транспортно-логистических услуг растет во всем мире, и в связи с этим решение различных задач маршрутизации становится все более актуальной проблемой. Основной целью в данных задачах является построение маршрутов для транспортных средств, обслуживающих определенное множество клиентов. При этом выбор неудачного маршрута влечёт за собой дополнительные издержки по транспортировке товара.

Класс задач маршрутизации в настоящее время является очень широким. Распространённым и самым ранним примером является задача коммивояжера или задача TSP (Travelling Salesman Problem). Для исследования была выбрана задача коммивояжера, зависящая от времени или задача TDTSP (Times Dependent Travelling Salesman Problem).

Алгоритмы, решающие данную задачу, разделяют на точные и эвристические. Так как задача коммивояжера принадлежит классу NP-трудных задач, очевидно, что точные алгоритмы применимы только для малоразмерных задач и не подходят для решения реальных задач. Поэтому, как правило, используют эвристические алгоритмы. Они генерируют решения, приближенные к оптимальному, но за меньшее по сравнению с точными методами время.

Особенностью многих эвристических алгоритмов является разнообразие решений, получаемых при применении алгоритма к одному и тому же примеру. Это обусловлено тем, что при построении маршрута различным вариантам продолжения маршрута соответствует вероятность их выбора, так как постоянный выбор наилучшего продолжения на каком-либо шаге алгоритма в итоге не всегда дает оптимальное решение. Ввиду больших расстояний, в рассматриваемых на практике задачах любое улучшение алгоритма позволит существенно уменьшить расхождение полученных решений с оптимальным.

В первой главе рассмотрены задачи TSP и TDTSP, представлены их математические модели. Во второй главе приведен обзор литературы по

природным алгоритмам и их сравнение. В третьей главе подробно рассмотрено применение муравьиного алгоритма (ACO) для построения решения задачи TDTSP. В четвертой дается определение динамической адаптации как критерия оценки алгоритмов и проведена оценка ACO. В пятой главе предложен метод динамической адаптации алгоритма, а так же его сравнение с ACO. В шестой главе описана проделанная работа и полученные результаты.

1.1. История задачи TSP

Откуда пошли истоки задачи коммивояжера, остаётся неясным. саму проблему и рекомендации, основанные на опыте для её решения, начинают упоминать в руководстве для коммивояжеров от 1832г. Также в нем рассматривают некоторые примеры маршрутов по Германии и Швейцарии, но там не содержится математической постановки задачи.

Первые упоминания в качестве математической задачи на оптимизацию принадлежат Карлу Менгеру, который сформулировал её на математическом коллоквиуме в 1930 году так: «Мы называем проблемой посыльного (поскольку этот вопрос возникает у каждого почтальона, в частности, её решают многие путешественники) задачу найти кратчайший путь между конечным множеством мест, расстояние между которыми известны» [1].

Современное название «Задача странствующего торговца» (Traveling Salesman Problem), предложил Хасслер Уитни из Принстонского университета.

В1954 году в институте RAND Corporation сформулировали задачу в виде задачи дискретной оптимизации и применили для её решения метод отсечений. Используя этот метод [2], они построили путь коммивояжера для одной частной постановки задачи с 49 городами и обосновали его оптимальность.

В 1972 году Ричард Карп доказал NP-полноту задачи поиска гамильтоновых путей, из чего, благодаря полиномиальной сводимости, доказывалась NP-трудность задачи коммивояжера. В марте 2005 года задача с 33 810 узлами была решена: был вычислен путь длиной в 66 048 945 и доказано отсутствие более коротких путей. В апреле 2006 было найдено решение для экземпляра с 85 900 узлами.

1.2. Постановка задачи TSP

Задача коммивояжёра может быть представлена в виде математической модели — графа $G=(V,A)$ [3]. Множество вершин $V=\{v_0,v_1,\ldots,v_{n-1}\}$ в графе соответствуют городам в задаче коммивояжёра, множество рёбер $A=\{(v_i,v_j)\,|\,v_i,v_j\in V, i\neq j\}$ – обозначают дороги, соединяющие пары городов. Каждое ребро из множества A имеет вес – d_{ij}. В зависимости от формулировки задачи, вес ребра (v_i,v_j) из графа G может означать время перехода, стоимость поездки или расстояние между соответствующими вершинами v_i и v_j .

Контур, включающий каждую вершину графа G хотя бы один раз, называется маршрутом коммивояжера. Так же контур, включающий каждую вершину графа G ровно один раз, называется гамильтоновым контуром. Тогда задачей коммивояжера называется задача поиска гамильтонова контура наименьшей общей длины.

Математическая модель задачи TSP

Пусть $I=\{0,\ldots,n-1\}$ — множество индексов вершин из тестовой задачи. Определим целевую функцию f как суммарное расстояние маршрута, покрывающего все вершины из рассматриваемой задачи.

Параметрами задачи являются элементы матрицы веса $D=\{d_{ij} \vee i,j \in I\}$. В случае асимметричной матрицы D задача коммивояжера моделируется ориентированным графом. То есть следует также учитывать, в каком направлении находятся ребра. В симметричном случае количество возможных маршрутов вдвое меньше асимметричного случая и выполняется равенство $d_{ij}=d_{ji}$. Дальше будем рассматривать задачи только с симметричной матрицей расстояний

Переменными задачи являются элементы бинарной матрицы переходов между вершинами $X=\{x_{ij} \vee i,j \in I\}$, которые равны 1, если в построенном маршруте для тестовой задачи присутствует ребро (v_i, v_j), 0 - иначе.

Тогда задача коммивояжёра может быть сформулирована и решена в рамках линейного целочисленного программирования. [4] [5] [6]. Требуется найти [7].

$$f=\sum_{i\in I}\sum_{j\neq i,\, j\in I} c_{ij}x_{ij} \to min$$

С учетом условий

$$\sum_{j\neq i,\, j\in I} x_{ij}=1, \forall i \in I \qquad (1)$$

$$\sum_{j\neq i,\, i\in I} x_{ij}=1, \forall j \in I \qquad (2)$$

$$v_i - v_j + n x_{ij} \leq n-1, 1 \leq i \neq j \leq n \qquad (3)$$

Ограничения (1) и (2) обеспечивают посещение города только один раз.

Неравенства (3) обеспечивают связность маршрута обхода городов, т.е. он не может состоять из двух и более не связных частей

1.3. Постановка задачи TDTSP

Задача TDTSP расширяет оригинальную задачу TSP, добавляя в неё параметр времени t, с изменением которого изменяются значения бинарной матрицы веса $D(t)=\{d_{ij}(t)|i,j\in I\}$.

Как и TSP, TDTSP определена на графе $G=(V,A)$, где $V=\{v_0,v_1,\dots,v_{n-1}\}$ – множество вершин и $A=\{(v_i,v_j)|v_i,v_j\in N, i\neq j\}$ множество ребер. Вершина v_0 - депо. Для каждого ребра (v_i,v_j) известно время $d_{ij}(t)$, затрачиваемое на его прохождение.

Таким образом, задача TDTSP сводится к нахождению кратчайшего маршрута, начинающегося в вершине v_0 и заканчивающегося там же, в котором коммивояжер проходит через каждую вершину один раз, и очередность прохождения влияет на остальное время пути.

Математическая модель задачи TDTSP

Параметрами задачи: параметр времени $t\in I$, элементы симметричной матрицы веса $D(t)=\{d_{ij}(t)|i,j\in I\}$.

Переменные модели: x_{ij}^t - принимает 1, если ребро (v_i,v_j) входит в маршрут, 0 - в обратном случае; t обозначает очередность прохождения рёбер.

Необходимо минимизировать целевую функцию [8]

$$f=\sum_{i\in I}\sum_{j\neq i,\,j\in I} d_{ij}(t)x_{ij}^t\to min$$

С учетом условий

$$\sum_{j\neq i,\,j\in I} x_{ij}=1,\forall i\in I \tag{4}$$

$$\sum_{j\neq i,\,i\in I} x_{ij}=1,\forall j\in I \tag{5}$$

$$v_i-v_j+n\,x_{ij}\le n-1,1\le i\neq j\le n \tag{6}$$

$$\sum_{i,j \in I} x_{ij}^{t} = 1, \forall t \in I$$

(7)

Ограничения (4) и (5) обеспечивают посещение города только один раз.

Ограничение (6) обеспечивают связность маршрута обхода городов, т.е. он не может состоять из двух и более не связных частей.

Ограничение (7) отвечает за то, что в любой момент времени t коммивояжёр движется только по одному ребру, что обеспечивает выполнения условия строго последовательного прохождения маршрута.

2. Обзор алгоритмов

Природно-естественные алгоритмы (Nature-inspired algorithms, NAs), черпают вдохновение из сложных, но хорошо организованных атрибутов природы. Природно-естественные алгоритмы стали очагом исследования из-за неэффективности точных алгоритмов (проблема увеличения размера задачи), таких как: линейное программирование [9], динамическое программирование [10], конечных элементов [11]. В общем NAs моделируют взаимодействие и иногда взаимозависимость природных элементов: растений, животных и рек. Самым популярным классом алгоритмов среди NAs является класс биологических алгоритмов. Однако некоторые NAs появились благодаря химии или физике. Они включают в себя Harmony Search (HS) алгоритмы, Intelligent Water Drop (IWD), имитации отжига (SA) и Black Hole (BH) [12]. В данном исследовании нас интересуют биологические алгоритмы.

2.1. Biology-inspired algorithms

Биологические алгоритмы (Biology-inspired algorithms,BAs) можно разделить на три больших класса, а именно: эволюционные алгоритмы (EvolutionaryAlgorithms, EAs) , которые основаны на принципах эволюций; алгоритмы роевого интелекта (Swarm Intelligence, SI), которые моделируют коллективное поведение у растений и животных; экологические алгоритмы, которые связаны с внешне- или внутривидовыми конкурентными процессами или кооперативным взаимодействием в пределах природных экосистем[13].

Эволюционные алгоритмы, как правило, моделируют итеративный прогресс, включающий в себя: рост, развитие, размножение, отбор и выживание, – как это наблюдается в популяциях животных. EAs создает допустимые решения, которые оптимизируются, переходя из поколения в поколение. К этой категории относятся: генетическое программирование (Genetic Programming, GP), алгоритм рисового поля (Paddy Field Algorithm,

PFA), эволюционная стратегия (Evolutionary Strategies, ES), генетический алгоритм (Genetic Algorithm, GA) и алгоритм Differential Evolution, (DE) [12]. Можно обратить внимание, что эксперты имеют различные мнения относительно классификации DE как эволюционного алгоритма. Причастие DE к эволюционным алгоритмам вытекает из использования им "Эволюции" в качестве одного из основного параметров (так же как и другими EAs). В текущее время весомые доводы не могут характеризовать класс DE как биологический алгоритм [14].

Вторая категория BAs – это алгоритмы роевого интеллекта (SI), которые касаются коллективно–социального поведения организмов. Основой для SI является коллективный интеллект групп простых агентов: насекомых, рыб, птиц, бактерий, червей и других животных, - воссозданный по анализу их поведения в реальной жизни. К примеру, у этих животных появляются исключительные навыки в разведке, когда они работают вместе как группа. Эти алгоритмы отслеживают коллективное поведение животных, которые проявляют децентрализованное поведение, самостоятельно организованные структуры при выполнении поиска пищи. Примерами таких алгоритмов являются: пчелиные колонии (the Bee Colony Optimization, BCO), алгоритм светлячка (Firefly Algorithm, FFA), метод роя частиц (Particle Swarm Optimization, PSO), муравьиный алгоритм (Ant Colony Optimization, ACO), алгоритм искусственных колоний пчел (Artificial Bee Colony, ABC), Bacteria Foraging Algorithm (BFA), и так далее [15].

Третья категория BAs – экологические алгоритмы, которые связаны с внешне- или внутривидовыми конкурентными процессами или кооперативным взаимодействием в пределах природных экосистем. Экосистема состоит из живых организмов вместе с их абиотической средой, с которой организмы взаимодействуют, такие как: вода, воздух и почва. Сотрудничество между организмами включает в себя разделение труда и представляет собой суть их социальности. Некоторые из взаимодействий являются кооперативными, другие конкурентоспособными. Они приводят к

сложной и гармоничной сбалансированности в экосистеме. Алгоритмы в этой категории являются PS2O, основанные на методах сорняка и биогеографической оптимизации [16].

Рассмотрим несколько алгоритмов из класса SI более подробно.

2.2. Ant Colony Optimization (ACO)

Алгоритм ACO - это способ оптимизации на основе популяции, который был разработан Марко Дориго и успешно применяется для решения несколько NP-трудных задач комбинаторной оптимизации. Этот алгоритм был вдохновлен поведением колоний муравьев, особенно их поведением в процессе поиска пищи в реальной жизни. Обычно муравьи, когда покидают свои гнезда, начинают беспорядочно перемещаться вокруг районов, окружающих их гнезда в поисках пищи. Если какому-либо из муравьев попадается пища, он сначала собирает некоторые кусочки пищи и на своем пути обратно в гнездо распыляет химическое вещество под названием феромон как способ общения с его коллегами, что он нашел источник пищи. Другие близлежащие муравьи воспринимают аромат феромона и начинают двигаться в направлении феромонова следа. После того, как они обнаруживают источник пищи, они обновляют феромоный след для предупреждения других муравьев, возвращаясь, В течение времени несколько муравьев получают эту информацию и следуют по образовавшемуся пути

Еще одна интересная часть поведения муравьев заключается в том, что как только они возвращаются в гнездо, они оптимизируют свой маршрут. За короткое время муравьи создают более короткий маршрут к источнику пищи, чем предыдущие маршруты. Кроме того, если на более короткий маршрут поместить препятствие, делая движения невозможным, муравьи способны найти еще один короткий маршрут из доступных вариантов обходов препятствия.

Были различные модификации муравьиных алгоритмов, начиная с исходной Ant System (AS), переходя к Ant Colony System (ACS), затем Min-Max System Ant (MMAC), а затем Ant Colony Optimization (ACO) алгоритмам, и так далее [17]. В ACO колония муравьев на каждой итерации высчитывает вероятность, на основе которой муравей в узле i выбирает следующий узел j, к которому будет двигаться дальше. Выбор узла зависит от значения следа феромона $\tau_{ij}(t)$ и доступный эвристичестик η_{ij}. В TSP $\eta_{ij}=1/d_{ij}$. Так что муравей перемещается от места i к j с вероятностью

$$P_{ij}^{t}=\begin{cases}\dfrac{\left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum_{l\in N_{i^k}}\left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}, if\ j\in N_{i^k}\\ 0, иначе\end{cases}$$

Здесь $\tau_{ij}(t)$ представляет собой след феромона, η_{ij} представляет локальную эвристическую информацию, t представляет собой итерацию, N_{i^k} представляет собой множество узлов, в которые муравей может пойти, а α и β являются параметрами, которые управляют изменением следа феромона. К концу итерации след феромона на каждом ребре $i\,j$ будет модернизирована с помощью следующего уравнения:

$$\tau_{ij}(t+1)=(1-p)*\tau_{ij}(t)+\Delta\tau_{ij}^{best}(t) \tag{8}$$

В формуле (8), $\tau_{ij}(t+1)$ представляет собой след феромона в итерации $t+1$, параметр $\rho\in[0.1,0.9]$ отвечает за скорость испарения феромона. Количество феромона, оставленным лучшим муравьём, представлен в виде:

$$\Delta\tau_{ij}^{best}(t)=\begin{cases}\dfrac{1}{f\left(s^{best}(t)\right)}, & \text{если лучший муравей}\\ & \text{прошел в итерации } t \text{ по дуге } ij\\ 0, иначе\end{cases} \tag{9}$$

В (9), $f\left(s^{best}(t)\right)$ представляет собой стоимость лучшего решения $s^{best}(t)$.

2.3. Particle Swarm Optimization (PSO)

Метод роя частиц базируется на социальном поведение стаи птиц или косяке рыб, является одним из биологически-вдохновленных вычислительных методов, разработанным Эберхартом и Кеннеди [18]. Этот алгоритм получает решения, используя рой частиц, где каждая частица представляет собой вариант решения. По сравнению с эволюционно вычисляемой парадигмой, рой похож на население, и частица представляет собой индивидуума. В поисках решения частицы перелетают через многомерное пространство, где положение каждой частицы регулируется в соответствии с собственным опытом и информацией от соседей. Вектор скорости управляет процессом оптимизации, а также обрабатывает опыт каждой частицы и учитывает информацию в рамках ее окрестности. PSO имеет два управляющих уравнений для поиска решения - (10) и (11). Рассмотрим

$$v_{ij}^{t+1} = x\left(v_{ij}^{t} + \theta_1 U_1\left(b_{ij}^{t} - x_{ij}^{t}\right) + \theta_2 U_2\left(b_{|n|ij}^{t} - x_{ij}^{t}\right)\right) + \Delta\tau_{ij}^{best}(t)$$

(10)

Где v_{ij}^{t+1} представляет собой текущую скорость, v_{ij}^{t} - предыдущая скорость, x -фактор сужения, θ_1 и θ_2 коэффициенты ускорения, U_1 и U_2 - случайные числа, b_{ij}^{t} - наилучшее положение отдельных частиц, x_{ij}^{t} - текущее положение, и $b_{|n|ij}^{t}$ лучше положение роя. Следующее уравнение в PSO отвечает за положение роя

$$x_{ij}^{t+1} = x_{ij}^{t} + x_{ij}^{t+1}$$

(11)

В алгоритме PSO частицы движутся в области целевой функции $f:\Theta \in Rn$, где n представляет переменные, которые будут оптимизированы. Каждая частица на определенной итерации связана с тремя векторами:

(a) Текущая позиция, обозначается через x : Этот вектор записывает текущую позицию конкретной частицы.

(b) Текущая скорость, обозначается через v : Этот вектор сохраняет направление частицы поиска.
(c) Индивидуальное личное положение, обозначается через b : Этот вектор содержит наилучшее решение конкретной частицы до текущего времени, с того момента как начался поиск (момента начала выполнения алгоритма). В дополнение к этому, отдельные частицы связаны с лучшей частицей в рое, которую алгоритм PSO отслеживает на каждой итерации, чтобы помочь направить в область поиска перспективных решений [19].

2.4. Artificial Bee Colony

Этот алгоритм, который был вдохновлен поведением пчелиного роя при сборе меда, был предложен Karaboga и Akay в 2009 году [20]. Он ищет решения за счет использования трех классов пчел: разведчики, пчелы наблюдателей и рабочие пчелы. Разведчиками являются те, которые летают над пространством поиска в поиске решений (источник питания). Пчелы наблюдатели – это те, которые остаются в гнезде в ожидании доклада разведчиков, в то время как рабочие пчелы после просмотра танца пчел разведчиков присоединяются к уборке источника питания (эксплуатации). Особенность этого алгоритма заключается в его возможностях – трансформации пчел. Например, разведчик может преобразовать в рабочую пчелу, когда он (разведчик) участвует в уборке источника пищи, и наоборот.

Так правило, пчелы могут изменять свои статусы в зависимости от потребностей алгоритма в определенный момент времени. В этом алгоритме источником питания представляет собой решение задачи оптимизации. Объем нектара в качестве источника питания представляет собой качество (пригодность) решения. Кроме того, предполагает, что каждая из рабочих пчел использует только один источник пищи, а это означает, что число рабочих пчел также соответствует количеству источников пищи. Пчелы разведчики всегда ищут новые источники пищи $\vec{v_m}$ с более высоким

количеством нектара и/или качество $\vec{x_m}$ в близлежащем районе. Пчелы оценивают пригодность нектара, используя

$$v_{mi}=x_{mi}+\theta_{mi}(x_{mi}-x_{ki}),$$

(12)

где i это случайно выбранный индекс; θ_{mi} представляет собой случайное число в заданном диапазоне; x_{ki} - источником пищи. Качество решения $fit_m(\vec{x_m})$ рассчитывается по следующей формуле:

$$fit_m(\vec{x_m})=\begin{cases}\frac{1}{1+f_m(\vec{x_m})}x, if\ f_m(\vec{x_m})>0\\ 1+fit_m(\vec{x_m}), if\ f_m(\vec{x_m})<0\end{cases}$$

(13)

Из вышеизложенного ясно, что есть небольшое сходство (10) в PSO и (12) в ABC, так как каждый алгоритм вычитает переменные из лучших личных и индивидуальных частиц/пчел. Для PSO, вычитаемая переменная представляет нынешнее положение, а для ABC - это разведанное местоположение. Тем не менее, эти два уравнения отличаются в нескольких отношениях: в то время как PSO использует x (будучи фактором сужения) или ω (как фактор инерции, в некоторых версиях PSO), нет таких эквивалентов в ABC. К тому же, в то время как PSO использует случайные числа (U_1 и U_2), в ABC их не содержит, только обучаемые параметры. Так же, PSO использует коэффициенты ускорения (θ_1 и θ_2); в ABC их нет. Даже если ABC использует ту же методику поиска нахождений решений, алгоритмы совершенно различны.

2.5. Выводы

Для оптимизации поиска решений ACO использует технику строительства путей, в то время как PSO и ABC используют технику улучшения пути. PSO использует фон Неймана в качестве лучшей техники для распространения информации [21], а ACO получает хорошие результаты,

используя топологию кольца [22]. Топология фон Неймана позволяет частицам соединяться с соседними частицами на востоке, западе, севере и юге. Эффективность состоит в том, что конкретная частица взаимосвязана с четырьмя другими частицами, окружающими её. ACO использует топологию кольца. Конкретный муравей, распыляя феромон, изменяет текущее его количество на пути, по которому он идет, что в целом для системы обеспечивает быстрый и легкий поиск новых маршрутов.

3. Муравьиный алгоритм для задачи TDTSP

Выше был рассмотрен ACO для простой задачи коммивояжера. Хотя задача TDTSP является усложненным вариантом задачи коммивояжера, но алгоритм муравьиной оптимизации при применении к TDTSP изменений в структуре не произвел. Изменились только тип и представление начальных данных. Ниже представлено описание параметров для задачи TDTSP и основные функции модифицированного муравьиного алгоритма, который был предложен для решения задачи TDTSP в [23]. Значения параметров для алгоритма взяты из этой же статьи.

3.1. Параметры задачи TDTSP

Весовая матрица для задачи TDTSP обозначает время, которое требуется затратить, чтобы добраться от одного узла к другому. Изменение матрицы веса $D(t)$ задается формулой, где $d_{i,j}(n_{up})$ время пути из i узла в j для момента n_{up}, $d_{i,j}(0)$ означает оригинальные значения, которые даются для задачи TSP, t - текущее время, скобки [] обозначают целую часть, Δt обновляемый интервал времени, C_f коэффициент изменения дорожной ситуации $(0 \le C_f < 1)$, и $rand \in [-1,1]$ - рандомный параметр.

$$d_{i,j}(n_{up}(t)) = d_{i,j}(n_{up}(t)-1) * (1 + C_f * rand)$$

$$n_{up}(t) = [\frac{t}{\Delta t}]$$

В работе значения $d_{i,j}(n_{up}(t))$ рассчитываются заранее для всех $n_{up}(t)$, что ускоряет работу алгоритма. Когда коммивояжер покидает депо, параметр $t = 0$ и он обязательно должен вернуться в него.

3.2. Время пути маршрута

Время, необходимое для путешествия вокруг всего маршрута S, будет называться временем маршрута $T(S)$. Время путешествия между городами

рассчитывается с использованием времени t_i , когда коммерсант достигает одного из концов дуги. Время маршрута для решения S можно рассчитать по следующим формулам:

$$T(S)=\sum_{i=0}^{n-1} d_{i,i+1}\left(n_{up}(t_i)\right)$$

$$t_i=\begin{cases} 0, \wedge i=0 \\ t_{i-1}+d_{i-1,i}\left(n_{up}(t_{i-1})\right), \wedge \textit{иначе} \end{cases}$$

3.3. Жадный алгоритм

Жадный алгоритм [24] широко используется в генерации начальных решений с нуля путем итеративного добавления компонентов решения в первоначально пустое решение до тех пор, пока решение не будет завершено. В TSP решение (тур) построен путем добавления города с самым маленьким значением времени пути. Алгоритмический план приведен ниже, где i, j и k являются номерами городов, S_p представляет собой частичное решение (то есть, список посещаемых городов), и N' представляет собой набор городов, который еще не посетили.

Алгоритм 1: Жадный алгоритм (next)

1: На входе получаем индекс следующего города

2: i=next;

3: $S_p=(0\,i)$; //первый город всегда депо

4 : $N'=N-(0)$;

5: Пока (N' не пустое){

6: $N'=N'-\{i\}$;

7: $k=arg\min_{j\in N'} d_{i,i+1}\left(n_{up}(t)\right)$;

8: $S_p=S_p+k$;

9: $i=k$;

10: }

11: Возвращаем полученный маршрут S_p.

3.4. Алгоритм локального поиска

Для улучшения уже построенного маршрута воспользуемся алгоритмом из класса итерационных эвристик. Он обрабатывает готовый маршрут, построенным один из конструктивных методов, в нашем случае муравьиным алгоритмом и жадным алгоритмом, последовательно улучшая его на каждой итерации. Один из самых популярных - 2-окружение (англ.: 2-opt[24]), который используется в основном алгоритме. Алгоритм 2-opt. удаляет два случайных несмежных ребра из маршрута и соединяет две получившиеся части маршрута другим способом. Существует единственный вариант для вставки двух новых рёбер, дабы полученное после произведенной операции решение было допустимым, т.е. являлось бы гамильтоновым циклом. Данные операции производятся при условии, что новый маршрут короче (быстрее) предыдущего. Пример операции 2-opt. продемонстрирован на Error: Reference source not foundError: Reference source not found.

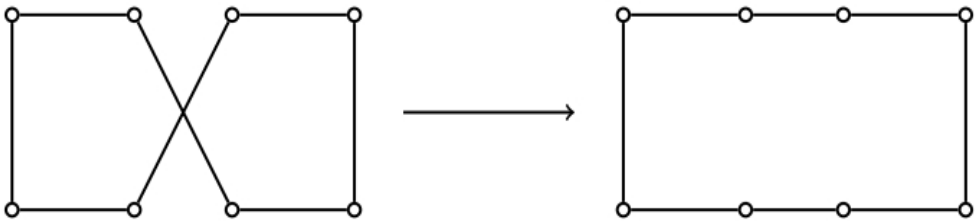

Рис 1. Пример операции 2-окружение

Алгоритм завершает свою работу, если не существует возможности улучшить готовый маршрут с помощью описанных операций.

3.5. Инициализация феромонов

Инициализирование уровня феромона происходит по следующей процедуре, где $S_{0k}, k=1, \acute{n}-1$ является решением, улучшенным за счет локального поиска (2-OPT) после того, как было построено с помощью жадного алгоритма, $r(0 \le r \le 1)$ параметр.

Алгоритм 2: Инициализация феромонов

1: От $i=1 \partial o i=n-1$ с шагом 1 запускаем Жад.алг.(i);

2: Улучшаем все полученные решения с помощью локальной оптимизации;

3 : $\tau_{i,j}=\tau_0, для \forall(i,j)\in A;$ //берется из формулы (14)

4: Обновляется $\tau_{i,j}$ по формуле (15);

$$\tau_{i,j}=\tau_0=\frac{n-1}{r\sum_{k=1}^{n-1}T(S_{0k})} \tag{14}$$

$$\tau_{i,j}=(1-r)\tau_{i,j}+r\frac{1}{n-1}\sum_{k=1}^{n-1}\delta\tau_{i,j}^{k} \tag{15}$$

$$\delta\tau_{i,j}^{k}=\begin{cases}\dfrac{1}{T(S_{0k})}, (i,j)\in S_{0k}\\ 0, иначе\end{cases}$$

Здесь первое слагаемое в правой части (15) означает равномерное испарение феромона, а второй член означает централизацию феромонов вокруг локальных оптимальных решений. Параметр r показывает отношение децентрализации инициализируемого феромонного уровня и централизации его. При $г=0$ предлагаемый способ похож на работу MMAS [25]. В противоположность этому, когда $г=1$, инициализируемый феромон распространяется только на дугах, содержащихся в локальных оптимальных решениях. Если локальный поиск не используется, предложенный метод со значением $r=0$ эквивалентен MMAS.

3.6. Основная процедура

Общая процедура модифицированного муравьиного алгоритма приведена ниже, где маршрут S_{ib} является лучшим решением в текущей итерации и маршрут S_{gb} является лучшим решением, найденным с начала алгоритма. Вероятность выбора следующего узла можно рассчитать по формуле (16).

$$p_{i,j}^{t}(t)=\begin{cases}\dfrac{\left[\tau_{i,j}(t)\right]^{\alpha}\left[\eta_{i,j}(t)\right]^{\beta}}{\sum_{l\in N_{i^k}}\left[\tau_{i,j}(t)\right]^{\alpha}\left[\eta_{i,j}(t)\right]^{\beta}}, if\ j\in N^k \\ 0, иначе\end{cases} \tag{16}$$

$$\eta_{ij}(t)=\frac{1}{d_{i,j}\left(n_{up}(t)\right)}$$

Алгоритм 3: Основная процедура

1: Считываем данные задачи TDTSP;

2: Задаем паратры для алгоритма;

3: Инициализируем феромон;

4: Запускаем цикл, который повториться q раз $\left(q<1+\frac{n}{100}\right)\{$

5: От $k=1 до k=n-1$ с шагом 1{

6: Строим решение S_k используя (16);

7: }

8: $S_{ib}=arg\ \min_{k=1,\acute{n}-1} T(S_k);$

9: Если ($T(\text{¿¿}ib)<T(S_{gb})$ то $S_{gb}=S_{ib};$

10: Обновляем фером исбользуя только глобальное решение;

11:}

12: Выдаем финально решение S_{gb} ;

---

4. Динамическая устойчивость

В настоящее время существуют различные критерии оценок эффективности эвристических алгоритмов. Один из них - это точность алгоритма. Для её определения алгоритм для каждого класса задач тестируется на тестовых примерах из открытых библиотек. Для задач с большой размерностью, в основном оптимальные решения не известны, это связано со сложностью расчетов точными алгоритмами. Поэтому для этих задач сохраняются лучшие решения (англ. Best known solution), полученные другими алгоритмами. Раздел с BKS постоянно обновляется.

Также важным критерием оценки алгоритма является скорость алгоритма. Методы теории сложности для оценки времени работы эвристических алгоритмов плохо применимы, причина тому их сложная структура. Поэтому под критерием сложности эвристических алгоритмов принимают непосредственное время работа алгоритма.

Время работы алгоритма очень сильно зависит от характеристик оборудования, на котором запускают алгоритм и само качество реализации кода алгоритма. Поэтому данный критерий почти не рассматривается при оценке эвристических алгоритмов.

Также известны такие критерии, как гибкость и простота [26]. Под гибкостью понимается способность алгоритма учитывать дополнительные ограничения и условия, которые принимаются к задаче. Пример гибкости очень хорошо наблюдается в ACO, когда реализация алгоритма для задачи TSP была перенесена на задачу TDTSP. Простота заключается в лёгкости программной реализации алгоритма. В работе [27] предложен алгоритм для нахождения решения транспортной задачи (Vehicle Routing Problem), который сам изменяет значения своих параметров в ходе поиска решения.

4.1. Динамическая устойчивость для эвристического алгоритма

В многошаговых динамических многоагентных играх есть свойство - сохранение оптимальности решения в каждой подзадаче с начальными условиями на оптимальной траектории, которая построена в начальный момент [28]. То есть у участников кооперации нет никакой объективной мотивации, следуя по выбранной траектории, чтобы отклоняться от неё в любой момент времени.

В отношении эвристических алгоритмов для задач TSP этот критерий именуется как динамическая устойчивость. Данное свойство для транспортной задачи означает, что данное решение сохраняет свойство оптимальности в каждой подзадаче при реализации начального маршрута. Это возможно только для оптимального решения, т.е. полученного точными методами. Эвристические алгоритмы, как правило, генерируют динамически неустойчивые решения.

Пусть S – произвольное решение задачи TDTSP. Разобьем его на две части: $\overset{+¿}{-¿ \cup S^{¿}}_{S=S^{¿}}$. Функция $T(S')$ принимает значения времени, затраченного на прохождение части S', входящую в маршрут S. Обозначим $\widetilde{V}$ – множество вершин маршрута $\overset{+¿}{S^{¿}}$ и пусть $\widetilde{v}_0$ —начальная вершина маршрута $\overset{+¿}{S^{¿}}$. Применим алгоритм для вершин из множества $\widetilde{V}$, где $\widetilde{v}_0$, v_{n-1} депо. Пусть $\tilde{S}$ – полученный маршрут. Теперь можно сформулировать определение динамической устойчивости решения.

Определение 1. Решение S, для которого выполняется $\begin{matrix} +¿ \\ S^{¿} \\ ¿ \\ T¿ \end{matrix}$ при любом разбиении на $\begin{matrix} +¿, \\ -¿uS^{¿} \\ S^{¿} \end{matrix}$ называется динамически устойчивым или состоятельным во времени.

4.2. Проверка уровня динамической устойчивости

Для описания определения экспериментального уровня динамической устойчивости алгоритма введем следующие обозначения: $P=\{p_1,p_2,\ldots,p_l\}$ – множество тестовых примеров; $R(p_i)=\{r_1(p_i),r_2(p_i),\ldots,r_m(p_i)\}-$ множество сгенерированных решений для тестового примера p_1 ; K – кол-во частей разбиения полученных решений; Q—количество запусков алгоритма проверки на динамическую устойчивость для одного тестового примера; $b(r_j(p_i))$— количество раз, когда решение r_i тестового примера p_i потеряло устойчивость при общем количестве запусков проверок Q.

Определение 2. Экспериментальным уровнем динамической устойчивости алгоритма будем называть величину, задаваемую формулой

$$¿R(p_i)\vee¿\begin{matrix} ¿R(p_i)\vee¿b(r_j(p_i)) \\ \sum_{j=1} \\ \frac{1}{¿} \\ ¿M\vee¿¿ \end{matrix}¿$$

$$con=1-\frac{1}{Q|M|}\sum_{i=1}^{¿}¿$$

Проверка решения на динамическую устойчивость происходит по следующему алгоритму:

Шаг1. $i=1, j=1, q=1.$

Шаг2. $k=1$.

Шаг3. $t=\frac{T(r_j(p_i))}{K}$, Множество $\tilde{V}$ – множество, состоящее из всех вершин маршрута $r_j(p_i)$, которые были пройдены после времени $t*k$,

соответственно $+¿(p_i)$ $r_j^{¿}$ – часть маршрута $r_j(p_i)$, которая была пройдена коммивояжером после времени $t*k$, Генерируем решение $\tilde{r}_j(p_i)$ для множества вершин $\tilde{V}$.

Шаг4. Если $+¿(p_i)$ $r_j^{¿}$ $¿$ $+¿(p_i)$ $r_j^{¿}$ $T¿$ не верно то решение неустойчиво, $q=q+1$, идем на **Шаг2**. иначе $k=k+1$, и проверяем Если $k \neq K$ то идем на **Шаг3**; иначе решение устойчиво, идем на **Шаг5.**

Шаг5. $q=q+1$ и если $q<Q$, переходим на **Шаг2.** Если $q=Q$, то $j=j+1, q=1$ и если $j<n$, то переходим на **Шаг2**. Если $j=n$, то $i=i+1, j=1, q=1$ и если $i<¿P \vee ¿$, то переходим на **Шаг2**, иначе СТОП.

Не трудно заметить, что кол-во динамически неустойчивых решений будет меньше, чем общее кол-во запусков алгоритма. Из этого следует, что оценка $con \leq 1$, и чем выше её значение, тем выше уровень динамической устойчивости и лучше работает исследуемый алгоритм.

4.3. Результаты проверки на динамическую устойчивость

Для проверки уровня динамической устойчивости муравьиного алгоритма для задачи TDTSP были взяты 5 тестовых задач из стандартной библиотеки Travelling Salesman Problem Library [29]. Значение параметров для: |P|=5, M=10, K=5, R=25. Результаты приведены в таблице1.

Для каждой из задач было сгенерировано по 25 различных решений. Для каждого решения проводилось 10 тестов. Исходный маршрут разбивался на 5 периодов. Если решение становилось в ходе проверки динамически неустойчивым, то текущий тест прекращался, а результат фиксировался. После завершения пятого периода проводить эксперименты не имеет смысла, т.к. маршрут обхода городов будет закончен.

задача		Кол-во дин. неус. решений	

	Количество экспериментов	1 период	2 период	3 период	4 период	Кол-во дин. уст. решений
Eil51	250	120	70	20	3	37
Eil76	250	147	43	21	2	37
Eil101	250	110	58	24	5	53
KroA200	250	90	32	11	3	114
Lin318	250	78	28	8	1	135
Сумма	1250	545	231	84	14	376

Таблица 1 Оценка динамической устойчивости ACO для TDTSP

Количество экспериментов, в которых начальное решение является динамически устойчивым после всех периодов, отличается для каждой тестовой задачи. Это обусловлено уникальной топологией исходных данных в задачах. Стоит заметить, что с увеличением размерности задачи коэффициент динамической устойчивости также увеличивается. Среднее значение экспериментального уровня динамической устойчивости муравьиного алгоритма для решения задачи Time-Depend Travelling Salesman Problem равно 0,6504

$$con\,ACO = 0{,}6504$$

Данное значение является довольно низким: только четверть сгенерированных на начальном этапе решений сохраняют свойство оптимальности в процессе своей реализации. Это означает, что существуют другие маршруты, которые можно получить динамически с помощью того же эвристического алгоритма, значения целевой функции которых будет меньше по сравнению с начальным решением. Алгоритм динамического улучшения эвристических алгоритмов будет подробнее рассмотрен в следующей главе.

5. Динамическая адаптация

Используя определения динамической устойчивости и проанализировав полученные данные в предыдущей главе, замечаем, что если после периода k начальное решение не соответствует критерию динамической устойчивости, т.е. для оставшейся части узлов было найдено решение лучше, то коммивояжеру нужно продолжить движение по нему. Таким образом, данные действия обеспечат то, что суммарная стоимость итогового маршрута будет меньше изначального. Сама идея постоптимизации была рассмотрена в статье [30], в ней рассматривалась идея сохранения оптимальности решения, когда после получения начального оптимального решения в задачу случайным образом добавлялся или убирался узел. Сама область, на которой базируется данная адаптация эвристического алгоритма, широко исследуется в области теории игр [28][31][32].

5.1. Динамическая адаптация для ACO

Динамическая адаптация(DA) эвристического алгоритма заключается в следующем:

Генерируем N решений с помощью алгоритма ACO для всех узлов, из них выбираем лучшее. Это и будет начальным решением.

Маршрут разбивается на K частей.

После прохождения каждой части, для оставшихся узлов генерируем N решений с помощью той же эвристики. Если среди них есть решение лучше, чем текущее, то заменяем оставшийся маршрут на него, иначе следуем текущему маршруту.

Данный алгоритм в основном будет генерировать решения лучше, чем простой муравьиный алгоритм. Это связано с тем, что: 1) каждый раз уменьшая размерность задач, увеличивается точность алгоритма; 2) каждая новая подзадача отличается от предыдущей своим суммарным

расположением узлов, что может поспособствует к нахождение локальных или глобальных оптимумов.

5.2. Результаты применения динамической адаптации

Чтобы показать качество работы динамической адаптации муравьиного алгоритма, проведем сравнение его решений с решением простого муравьиного алгоритма. Для этого запустили оба алгоритма для тестовых задач по 20 раз, каждый запуск выдавал 50 решений. Параметры для запуска DAACO: K=5, R=50. В Таблица 2. приведены средние значения полученных решений, а также показаны лучшее решений для тестовых задач.

задача	Среднее значение		Лучшее значение	
	ACO	DAACO	ACO	DAACO
Eil51	461,26	435,04	426	395
Eil76	576,21	551,43	538	514
Eil101	673,64	634,73	629	612
KroA200	31379,05	30340,80	29368	29002
Lin318	45315,16	44948,71	42029	41950

Таблица 2.Сравнение ACO и DAACO

Так как алгоритм динамической адаптации почти на каждом шаге изменял изначальный маршрут, то это порождает вопрос, насколько динамически устойчивыми являются решения, полученные алгоритмом DAACO. Эксперименты проведем при тех же условиях, как и для алгоритма ACO. Значения параметров: |P|=5, M=10, K=5, R=25. Результаты отображены в Таблица 3.

задача	Количество экспериментов	Кол-во дин. неус. решений				Кол-во дин. уст. решений
		1	2	3	4	
Eil51	250	16	5	1	0	228
Eil76	250	13	6	0	0	231
Eil101	250	20	9	2	0	219
KroA200	250	8	2	0	0	240
Lin318	250	12	4	0	0	234
Сумма	1250	69	26	3	0	1152

Таблица 3. Проверка на дин. уст. DAACO

Таким образом, экспериментальный уровень динамической устойчивости алгоритма DAACO для задачи TDTSP равен 0,9608

$$conDAGA = 0{,}9608$$

5.3. Анализ результатов

Из данных в Таблица 2 видно, что среднее значение генерируемых решений алгоритмом DAACO меньше, чем алгоритмом ACO.

Вычислим процент улучшения по формуле

$$k = \frac{T_{ACO}(p_i) - T_{DAACO}(p_i)}{T_{ACO}(p_i)} 100$$

Где $T_{ACO}(p_i)$ — время маршрута, полученного муравьиным алгоритмом для задачи p_i, $T_{DAACO}(p_i)$ – время маршрута полученного динамически адаптированным муравьиным алгоритмом.

Результаты видны в Таблица 4.

задача	коэффициент улучшения	
	Среднее значение	Максимальное значение
Eil51	5,68	7,28
Eil76	4,30	4,46
Eil101	5,78	2,70
KroA200	3,31	1,25
Lin318	0,81	0,19

Таблица 4. Коэффициент улучшения

Средний процент улучшения решений равен 4%.

Из анализа всех данных следует, что динамическая адаптация муравьиного алгоритма позволяет получать лучшие решения относительно критерия динамической устойчивости и точности, чем простой муравьиный алгоритм.

6. Заключение

В работе была исследована одна из задач маршрутизации транспорта – динамическая задача коммивояжера. Были рассмотрены различные эвристические алгоритмы природного класса.

Подробно изучен, описан и реализован на языке программирования Java муравьиный алгоритм (ACO) для решения задачи TDTSP. Произведена оценка уровня динамической устойчивости решений, генерируемых данной эвристикой. Из-за того, что эвристики не обеспечивают получение оптимальных решений, но близких к ним, среднее значение этой величины для рассмотренных задач получилось равным 0,6504.

Принцип динамической адаптации показал, что любой алгоритм, который генерирует различные решения для одной и той же задачи, можно улучшить с помощью алгоритма динамической адаптации. Динамическая адаптация муравьиного алгоритма (DAACO) повысила уровень динамической устойчивости решений до 0,9608. Также среднее значение длины решений, сгенерированных алгоритмом DAACO, получилось меньше, чем классическим муравьиным алгоритмом. Средний процент улучшения решений равен 4%. Результаты экспериментов показали, что использование динамической адаптации муравьиного алгоритма будет эффективней, чем простого муравьиного алгоритма, за счет генерируемых маршрутов меньшей длины.

7. Список литературы

1. Schrijver A. On the history of combinatorial optimization (till 1960) // Handbook of Discrete Optimization. 2005. P. 1–63
2. Vasek Chvatal, William J. Cook, George B. Dantzig, Delbert Ray Fulkerson,and Selmer M. Johnson. Solution of a large-scale traveling-salesman problem. In 50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art, pages 7–28. 2010.
3. Майника Э. Алгоритмы оптимизации на сетях и графах. М: Мир, 1981. 323 с.
4. Christos H. Papadimitriou and Kenneth Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.
5. George B. Dantzig. Linear programming and extensions. Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ, 1963.
6. Blum C., Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surv. 2003, Vol. 35, No 3. P.268–308.
7. Miller C. E., Tucker A. W., Zemlin R. A. Integer programming formulation of traveling salesman problems // J. ACM. 1960. Vol. 7, No 4. P. 326–329.
8. H. Abeledo, R. Fukasawa, A. Pessoa, and E. Uchoa, The Time Dependent Traveling Salesman Problem: Polyhedra and Algorithm, Mathematical Programming Computation 5 (2013), 27-55.
9. Ben-Israel, “A Newton-Raphson method for the solution of systems of equations,” Journal of Mathematical Analysis and Applications, vol. 15, pp. 243–252, 1966.
10. D. P. Bertsekas, Dynamic Programming and Optimal Control,vol. 1, Athena Scientific, Belmont, Mass, USA, 1995.
11. R. G. Ghanem and P. D. Spanos, Stochastic Finite Elements: A Spectral Approach, Courier Corporation, 2003.
12. S. Binitha and S.S. Sathya, “A survey of bio inspired optimization algorithms,”

International Journal of Soft Computing and Engineering, vol. 2, pp. 137–151, 2012.

13. F. Dressler and O. B. Akan, “A survey on bio-inspired networking, ”Computer Networks, vol. 54, no. 6, pp. 881–900, 2010.
14. X.-S. Yang and S. Deb, “Two-stage eagle strategy with differential evolution,” International Journal of Bio-Inspired Computation, vol. 4, no. 1, pp. 1–5, 2012.
15. S. K. Dhurandher, S. Misra, P. Pruthi, S. Singhal, S. Aggarwal, I. Woungang, “Using bee algorithm for peer-to-peer filesearching in mobile ad hoc networks,” Journal of Network and Computer Applications, vol. 34, no. 5, pp. 1498–1508, 2011.
16. A.R. Mehrabian and C. Lucas, “A novel numerical optimization algorithm inspired from weed colonization,” Ecological Informatics, vol. 1, no. 4, pp. 355–366, 2006.
17. M.Dorigo, G. Di Caro, and L.M. Gambardella, “Ant algorithms for discrete optimization,” Artificial Life, vol. 5, no. 2, pp. 137–172, 1999.
18. M. Clerc, Particle Swarm Optimization, vol. 93, John Wiley & Sons, 2010.
19. Z. Yuan, M. A.M. deOca, M. Birattari, T. Stutzle, “Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms,” Swarm Intelligence, vol.6, no. 1, pp. 49–75, 2012.
20. D. Karaboga, B. Akay, “A survey: algorithms simulating bee swarm intelligence,” Artificial Intelligence Review, vol. 31, no. 1–4, pp. 61–85, 2009.
21. J. Kennedy and R. Mendes, “Population structure and particle swarm performance,” in Proceedings of the Congress on Evolutionary Computation (CEC ’02), vol. 2, pp. 1671–1676, IEEE, Honolulu, Hawaii, USA, May 2002.
22. A.M. Mora, P. Garcıa-Sanchez, J.J. Merelo, and P. A. Castillo, “Migration study on a pareto-based island model for MOACOs,” in Proceedings of the 15th Genetic and Evolutionary Computation Conference (GECCO ’13), pp. 57–64, July 2013.
23. Hitoshi K., Junichi O. Solving Time-Dependent Traveling Salesman Problems Using Ant Colony Optimization Based on Predicted Traffic // Advances in

Intelligent and Soft Computing. 2012. 151.
24. Dorigo, M., Stutzle, T. Ant colony optimization. The MIT press, 2004.
25. Stutzle, T., Hoos, H. H. MAN-MIN ant system. Future Generation Computer System, Vol. 16, No. 8, 2000, 889–914.
26. J.F. Cordeau, M. Gendreau, G. Laporte, J.Y. Potvin, and F.Semet. A guide to vehicle routing heuristics. Journal of the Operational Research Society, pages 512–522, May 2002.
27. .Jean-Francois Cordeau, Michel Gendreau, and Gilbert Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks, 30(2):105–119, 1997.
28. Петросян Леон Аганесович и Зенкевич Николай Анатольевич. Принципы устойчивой кооперации. Управление большими системами: сборник трудов, (3):100–120, 2009.
29. G. Reinelt. Travelling Salesman Problem Library, 2008. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.
30. Giorgio Ausiello, Bruno Escoffier, JeRoMe Monnot, and Vangelis Paschos. Reoptimization of minimum and maximum traveling salesman's tours. J. of Discrete Algorithms, 7(4):453–463, December 2009.
31. .V. Zakharov and M. Dementieva. Multistage cooperative games and problem of time consistency. International Game Theory Review, 6:157–170, 2004.
32. .V.V. Zakharov and A.N. Shchegryaev. Multi-period cooperative vehicle routing games. Contributions to Game Theory and Management, 7(2):349–359, April 2014.