САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ ЭНЕРГЕТИЧЕСКИХ СИСТЕМ

**Корнилов Вадим Юрьевич**

 **Магистерская диссертация**

**Адаптация эвристических алгоритмов маршрутизации на большой сети**

Направление 01.04.00

Прикладная математика и информатика

Магистерская программа «Математическое и

информационное обеспечение экономической деятельности»

Научный руководитель,

доктор физ.-мат. наук,

профессор

Захаров В. В.

Санкт-Петербург

2018

Оглавление

[1. Введение 3](#_Toc514498942)

[1.1. Виды задачи VRP 4](#_Toc514498943)

[1.2. Постановка задачи СVRP 5](#_Toc514498944)

[1.3. Математическая модель задачи CVRP 6](#_Toc514498945)

[2. Муравьиный алгоритм для задачи VRP 9](#_Toc514498946)

[2.1. Муравьиная эвристика 9](#_Toc514498947)

[2.2. ACO для задачи CVRP 13](#_Toc514498948)

[2.3. Используемая реализация ACO 13](#_Toc514498949)

[2.4. Основная процедура ACO 15](#_Toc514498950)

[2.5. Выбор параметров для ACO 15](#_Toc514498951)

[3. Динамическая устойчивость 17](#_Toc514498952)

[3.1. Динамическая устойчивость для эвристического алгоритма 18](#_Toc514498953)

[3.2. Проверка уровня динамической устойчивости 19](#_Toc514498954)

[3.3. Результаты проверки на динамическую устойчивость 20](#_Toc514498955)

[4. Динамическая адаптация 22](#_Toc514498956)

[4.1. Динамическая адаптация для ACO 22](#_Toc514498957)

[5. Оценка вычислительной сложности алгоритма DAACO 24](#_Toc514498958)

[6. Анализ эффективности DAACO 26](#_Toc514498959)

[6.1. Оценка динамической устойчивости 26](#_Toc514498960)

[6.2. Сравнение решений DAACO и ACO 27](#_Toc514498961)

[6.3. Эффективность DAACO за время работы ACO 28](#_Toc514498962)

[7. Заключение 29](#_Toc514498963)

[8. Список литературы 31](#_Toc514498964)

[Приложение 1 33](#_Toc514498965)

[Приложение 2 34](#_Toc514498966)

[Приложение 3 38](#_Toc514498967)

[Приложение 4 42](#_Toc514498968)

[Приложение 5 43](#_Toc514498969)

# Введение

Транспортная логистика занимает важное место на современном рынке, предприятия должны ориентироваться на интересы потребителей, а не на собственные. Для этого они должны сконцентрироваться на максимальном удовлетворении запросов клиентов. Оно выражается в достойном качестве товаров и услуг. Но больше всего влияют на цену товаров логистические издержки. В частности, поставщики поставляют ресурсы производителю, производитель направляет готовый товар посредникам, а те, в свою очередь – конечному покупателю. Любое предприятие заинтересовано в максимально продуманном, быстром и дешевом перемещении продукции.

Есть большое разнообразие математических моделей для описания логистических процессов. Классической задачей является проблема маршрутизации транспортного средства (Vehicle Routing Problems, VRP) – это общее название для целого класса задач, в которых для нескольких разбросанных городов или клиентов должен быть определен набор маршрутов для транспортных средств, расположенных на одном или нескольких складах. Данная модель предложена Данцигом и Рамсером в 1959 году.

VRP относится к классу NP-полных задач, что накладывает ограничение размерности задачи для целесообразной возможности применения точных алгоритмов. Для решения задачи на больших сетях используют алгоритмы, основанные на различных эвристиках и в том числе их комбинации. Но из-за особенности в вычислениях для одной и той же задачи эвристические алгоритмы могут генерировать различные решения, притом в основном лишь приближенные к оптимальному решению. В связи с этим присутствует большая заинтересованность в модификациях эвристических алгоритмов для их улучшения. У каждой реальной задачи имеются свои особенности, для которых нужны оптимизации вычислительных алгоритмов по определенным параметрам. Например, для курьерской службы важно время построения маршрутов, так как заказы могут поступать в реальном времени и есть необходимость в моментальном пересчете маршрута, в то же время для крупных предприятий важно построение маршрутов с низкой стоимостью, на которое они могут отвести определенное время, но гарантировать надежность данных маршрутов.

В первой главе рассмотрены задачи VRP и CVRP, представлена математическая модель задачи CVRP. Во второй главе подробно рассмотрен муравьиный алгоритм (ACO) и способы его применения для задачи CVRP. Проведен эксперимент по определению лучших наборов параметров для алгоритма ACO. В третьей главе дается определение динамической адаптации как критерия оценки алгоритмов и проведена оценка ACO. В четвертой главе предложен метод динамической адаптации алгоритма, приведена блок-схема алгоритма. В пятой главе рассмотрена вычислительная сложность алгоритма ACO и проведено вычисление её вида для алгоритма DAACO. В шестой главе произведен анализ алгоритма DAACO, метод экспериментального сравнения с алгоритмом ACO на ряде тестовых задач с использованием различных критериев. В седьмой главе описана проделанная работа и полученные результаты с выводами.

## Виды задачи VRP

Первое появление задачи транспортной маршрутизации появилось в статье [1], где в 1959 году была поставлена математическая формулировка для поставки бензина на заправочные станции и представлено её решение. Сейчас эта задача является одной из самых известных в области комбинаторной оптимизации. Общий смысл задачи VRP представляется следующим образом: m транспортных средств, находящихся в депо, должны доставить товар n клиентам с минимальными затратами на перевозку товара. Решением является множество замкнутых маршрутов в депо, удовлетворяющие условию, что все клиенты посещались только один раз. Затраты на перевозку оптимизируются за счет уменьшения количества транспортных средств или оптимизации построенных маршрутов.

Для большинства задач транспортной маршрутизации вводят дополнительные ограничения, их можно классифицировать, как это сделано в статье [2], часто используемы из них:

1. Capacitated Vehicle Routing Problems (CVRP). Все транспортные средства имеют фиксированную вместимость.
2. Vehicle Routing Problems with Time Windows (VRPTW). Каждому клиенту задан определенный временной интервал (окно), в течение которого ему необходимо доставить груз.
3. [Multiple Depot Vehicle Routing Problems (MDVRP).](http://rain.ifmo.ru/cat/view.php/theory/unsorted/vrp-2006#mdvrp) Указано больше одного депо, которые могут использоваться для обслуживания клиентов.
4. Periodic Vehicle Routing Problem (PVRP). Задачи с возможностью доставки грузов в течение длительного времени. Общий период доставки расширяется до нескольких дней, по истечении которых каждый клиент должен быть посещен как минимум один раз. К тому же, транспортное средство может вернуться в депо в следующие дни после отправления.
5. Stochastic Vehicle Routing Problem (SVRP). В данной задаче для описания количества клиентов, их запросов или длины пути могут быть использованы случайные переменные. Таким образом, во время решения эти параметры могут измениться.

## Постановка задачи СVRP

Модель задачи представляется в виде графа – полный направленный граф, где – множество узлов, причем узел 0 представляет депо для транспортных средств с одинаковой емкостью Q и n узлами, которые представляют географически рассредоточенных клиентов. Матрица описывает множество дуг, транспортную сеть между узлами. Каждый клиент имеет определенный положительный спрос Стоимость перемещения связана с каждой дугой Матрица стоимости симметрична, , для всех и удовлетворяет условию неравенства треугольника для всех [3]. Минимальное необходимое количество транспортных средств для обслуживания всех клиентов вычисляется как .

На Рис 1 показан пример возможного решения задачи CVRP с 7 клиентами и 3 транспортными средствами емкостью . Требование клиентов указано рядом с узлами.



Рис 1. Граф задачи CVRP, n = 7, Q = 50.

## Математическая модель задачи CVRP

Бинарная переменная определяет перемещение транспортного средства по дуге в решении задачи. Целочисленная модель линейного программирования задачи CVRP [4] может быть представлена как:

Минимизация целевой функции:

 (1)

С учетом условий:

 (2)

 (3)

 (4)

 (5)

 (6)

 (7)

Целевая функция (1) минимизирует общую стоимость перевозок. Ограничение модели (2) обеспечивает, чтобы каждого клиента посетило ровно одно транспортное средство. Следующие ограничения (3) и (4) гарантируют, что каждое транспортное средство может покинуть депо только один раз, а количество транспортных средств, посещающих каждый узел, равно количеству покидающих его. В (5) задаются ограничения вместимости транспортного средства, удостоверяющие, что сумма требований клиентов, принадлежащих маршруту транспортного средства, меньше или равна емкости транспортного средства, следующего по этому маршруту. Ограничение (6) отклоняет решения, которые не содержат циклов, исключающих депо. Оставшееся обязательное ограничение (7) определяет области определения переменных. Эта модель известна как a three-index vehicle flow formulation. Число неравенств для исключения циклов без депо увеличивается экспоненциально с увеличением количества узлов.

# Муравьиный алгоритм для задачи VRP

## Муравьиная эвристика

Создание алгоритма муравьиной колонии (Ant Colony Optimization) [5] было вдохновлено поведением реальных муравьев. Муравьи случайно исследуют окрестности муравейника, когда они находят пищу, они возвращаются в муравейник, оставляя маршрут из феромонов, химического вещества, которое может восприниматься другими муравьями. Муравьи могут следовать различными путями к источнику питания и обратно, но было замечено [6], что благодаря последовательным проходам муравьев по маршрутам и, следовательно, усилению феромонов на них, остается только самый короткий путь, так как муравьи предпочитают следовать по маршрутам с более сильной концентрацией феромонов, а феромон со времен испаряется. Со временем большинство муравьев будут путешествовать по пути с наибольшей концентрацией феромона, в то время как меньшинство все равно будет выбирать альтернативные пути. Поведение этого меньшинства важно, поскольку оно позволяет исследовать окружающую среду, чтобы найти еще лучшие решения, которые изначально не учитывались. Поэтому выбор пути является вероятностным, и, хотя на него сильно влияет интенсивность феромонов, он по-прежнему допускает случайные отклонения от текущего наилучшего решения.

Алгоритм ACO воспроизводит это поведение, добавляя некоторые функции, чтобы сделать его более эффективным при реализации на компьютере. Муравьи представляют собой набор агентов. Они ищут решение, посещая ряд узлов на графике. Их выбор следующего узла для посещения определяется двумя параметрами: тропой перемещения и привлекательностью самого пути. Как и реальные муравьи, искусственные муравьи предпочитают в большинстве случаев выбирать маршрут, основанный на выборе пути с самой сильной концентрацией феромона и наивысшей привлекательностью. Тем не менее, во многих случаях выбор будет сделан случайно.

Привлекательность перехода от узла к вычисляется в соответствии с эвристикой, она выражает естественное предпочтение выбора маршрута. В задаче кратчайшего пути привлекательность обычно равна обратной к величине расстояния; в других вариантах VRP привлекательность может также зависеть от других параметров, помимо расстояния, например, в VRPTW она также зависит от текущего времени и границ временного окна клиентов, которые будут посещаться.

Уровень тропы зависит от уровня феромонов и представляет собой динамический индикатор полезности. Другими словами, если муравей видит тропу с высоким уровнем феромона, ведущую к узлу, он знает, что это перспективное направление для изучения. Когда процедура поиска завершена, и муравьи вычислили набор решений, то тогда информация о феромоне, связанная с некоторыми ребрами , обновляется в соответствии со следующим уравнением:

 (8)

где – параметр, отвечающий за степень испарения феромона, а – его прирост, которые зависят от конкретной реализации ACO. Феромон также испаряется, чтобы избежать попадания в локальные минимумы. Испарение уменьшает феромон по всем тропам итерация за итерацией, обычно путем экспоненциального распада.

Существует несколько метаэвристических реализаций ACO, которые отличаются тем, как используется и обновляется феромон. Первой реализацией является Ant System [7]: когда муравей находится в узле , следующий узел выбирается вероятностно согласно произвольно-пропорциональному правилу:

(9)

Где – вероятность перехода из в , а – множество узлов, которые можно посещать из . Параметры и определяют влияние феромона и видимости привлекательного пути.

Муравьи строят свои решения параллельно. В конце каждой процедуры поиска (итерации) весь набор вычисленных решений используется для обновления тропы феромонов. В этом случае уравнения (8) вычисляется по следующей формуле:

где – количество муравьев, а - количество феромона, оставляемого на дуге муравьем , которое обычно вычисляется как , если муравей использовал ребро в своем маршруте, – значение целевой функции для маршрута k-ого муравья, а – постоянный параметр силы феромона.

Первоначально Ant System была применена к решению проблемы Travelling Salesman, но не смогла конкурировать с современными алгоритмами в этой области. Чтобы улучшить систему AS, Гамбарделла и Дорииго предложили в 1995 году алгоритм Ant-Q [8], а в 1996 году – Ant Colony System (ACS) [9][7], упрощенная версия Ant-Q, которая поддерживала примерно такой же уровень эффективности, измеряемый в сложности алгоритма и в вычислительных результатах. ACS представила концепции локального и глобального обновления и правила псевдослучайно-пропорционального перехода, которые лежат в основе любой успешной реализации ACO.

В ACS в конце каждой итерации глобальное обновление использует только лучшее решение, которое определили на текущий момент, для обновления тропы феромонов. Единственными ребрами, которые изменяются, являются те ребра , которые принадлежат наилучшему решению со значением целевой функции . Эта стратегия сокращает время до схождения алгоритма, напрямую концентрируя поиск в окрестности наилучшего решения. Формула глобального обновления:

Локальное обновление выполняется гораздо чаще: каждый раз, когда муравей решает использовать ребро в своем решении, феромон изменяется:

где – начальное значение феромона, определяемое как , – количество клиентов и – длительность тура, полученная путем выполнения одной итерации ACS без компонента феромона (это эквивалентно вероятностной эвристике ближайшего соседа).

Во время построения нового решения вводится правило псевдо-случайно-пропорционального перехода. Это правило является компромиссом между правилом выбора псевдо-случайного состояния, обычно используемого в Q-learning [10], и правилом выбора произвольно-пропорционального действия, обычно используемого в Ant System. Оно приведено в уравнении (9). С псевдо-случайным правилом, если – случайная величина, равномерно распределенная на отрезке , с вероятностью (эксплуатация), следующим выбранным узлом является тот, у которого наилучшее значение , а с вероятностью (разведка) узел выбирается с использованием правила случайного пропорционального выбора Ant System. Таким образом, псевдослучайное правило может быть выражено следующим образом:

где – параметр , обычно равный 0,9. Псевдослучайное правило обеспечивает простой способ баланса между разведкой новых состояний и использованием накопленных знаний.

Было доказано, что ACS очень эффективен при решении многих задач маршрутизации графов. Как и в основе ACO, ACS использует фоновые действия для выполнения глобальных процессов, которые невозможно применить для искусственных муравьев, которые взаимодействуют только локально. Типичным действием фонового процесса является запуск локального поиска, чтобы улучшить текущее лучшее решение, найденное ACO в текущей итерации.

Различные алгоритмы, реализующие разные версии ACO, и вид задачи VRP также оказывают значительное влияние на реализацию ACO.

## ACO для задачи CVRP

CVRP является основным вариантом VRP, и он первым привлек внимание исследователей, пытающихся применить метаэвристику ACO к VRP. Один из первых подходов – Bullnheimer et al. [11], [12], где целевая функция имеет уникальную цель минимизировать общую длину маршрута.

Чтобы решить эту проблему, муравьи строят маршруты, последовательно выбирая клиентов для посещения, пока каждый клиент не будет обслужен. Если выбор клиента приводит к нарушению ограничения на вместимость транспорта, муравей возвращаются в депо.

## Используемая реализация ACO

Для исследования был реализован метаэвристический алгоритм ACO с определенными особенностями.

Все муравьи для выбора направления перемещения подчиняются произвольно-пропорциональному правилу (9), но из-за ограничения вместимости транспортного средства, при каждом посещении клиента объем текущего груза в транспорте уменьшается, следовательно, появляются клиенты, потребности которых уже нельзя удовлетворить в текущем состоянии. Правило выбора для используемой реализации будет иметь вид:

 (10)

где – множество узлов, которые можно посещать из и (текущий объем груза).

Начальный уровень феромона для тропы задается на основе матрицы стоимости:

Обновление феромона будет включать как локальную процедуру, так и глобальную. Во время локальной процедуры феромон будет обновляться только на тех ребрах, по которым прошел муравей, то есть принадлежавшим найденному им решению Формула локального обновления феромона для k-ого муравья:

 (11)

Во время глобального обновления феромона будет происходить его испарение на всей тропе, интенсивность испарения зависит от параметра . Для лучшего решения на текущую итерацию к входящим в него ребрам добавляется феромон . Формула глобального обновления:

 (12)

Конечная формула обновления феромона в конце каждой итерации имеет вид:

## Основная процедура ACO

Общая процедура реализованного муравьиного алгоритма для задачи CVRP приведена ниже.

**Алгоритм 1:** Основная процедура

1: Считываем данные задачи CVRP;

2: Задаем параметры для алгоритма;

3: Инициализируем начальный уровень феромона;

4: Запускаем цикл, по количеству задананных итераций

5: Запускаем муравьев с начальными условиями{

6: Строим решение используя (10);

7: Производим локальное обновление феромона(11);

8: }

9:

9: Если то

10: Производим глобальное обновление феромона (12);

11:}

12:Выводим финально решение ;

## Выбор параметров для ACO

Для определения наилучших параметров были произведены тестовые запуски на задаче E-n22-k4.vrp из библиотеки [13], с перебором значений параметров . c шагом ; с шагом ; с шагом ; с шагом . Количество инициализированных муравьев соответствует количеству вершин в графе, для данной задачи , количество итераций . Для каждого набора параметров было произведено 5 запусков. В сумме необходимо произвести 73500000 запусков. Для уменьшения количества запусков и диапазона значений параметров вначале производился перебор параметров с шагом , что составило 1470000 генераций. Из них были составлены новые диапазоны значений параметров , , для которых были получены только оптимальные решения. Затем был произведен перебор на новых диапазонах параметров, но с изначальным шагом для них, что составило ещё 182000 запусков алгоритма. В итоговом анализе полученных данных [Приложение 1], для экспериментов был выбран набор параметров с максимальным значением параметра .

# 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 оптимальные решения в основном 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, причинной служит их сложная структура. Поэтому под критерием сложности эвристических алгоритмов понимают 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 [14]. 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 VRP изменяется под другие классы этой задачи. Простота заключается в легкости программной реализации алгоритма. В работе [15] предложен алгоритм для нахождения решения транспортной задачи (Vehicle Routing Problem), который самостоятельно изменяет значения своих параметров в ходе поиска решения.

## Динамическая устойчивость для эвристического алгоритма

В многошаговых динамических многоагентных играх есть такое свойство, как сохранение оптимальности решения в каждой подзадаче с начальными условиями на оптимальной траектории, которая построена в начальный момент [16]. То есть, у участников кооперации нет никакой объективной мотивации, чтобы отклониться от выбранной траектории, по которой они следуют, в любой момент времени.

В отношении эвристических алгоритмов для задач VRP 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 – произвольное решение задачи CVRP. Разобьем его на две части: . – целевая функция расстояния, затраченного на прохождение части , входящую в маршрут . Обозначим множество вершин маршрута , и пусть —начальная вершина маршрута . Применим алгоритм для вершин из множества . Пусть – полученный маршрут. Теперь можно сформулировать определение динамической устойчивости решения.

**Определение 1**. Решение , для которого выполняется при любом разбиении на называется динамически устойчивым или состоятельным во времени.

## Проверка уровня динамической устойчивости

Для описания определения экспериментального уровня динамической устойчивости алгоритма введем следующие обозначения: –множество тестовых примеров; множество сгенерированных решений для тестового примера ; – количество частей разбиения полученных решений; – количество запусков алгоритма проверки на динамическую устойчивость для одного тестового примера; количество раз, когда решение тестового примера потеряло устойчивость при общем количестве запусков проверок, равного .

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

Проверка решения на динамическую устойчивость происходит по следующему алгоритму:

**Шаг 1.**

**Шаг 2**..

**Шаг 3**. Множество – множество, состоящее из всех вершин маршрута , которые были пройдены после отметки , соответственно – часть маршрута , которая была пройдена коммивояжером после отметки . Генерируем решение для множества вершин .

˜

**Шаг 4**. Если неверно, то решение неустойчиво, , идем на **Шаг 2**. иначе , и проверяем: если , то идем на **Шаг 3**; иначе решение устойчиво, идем на **Шаг 5.**

**Шаг 5.** и если, переходим на **Шаг 2.** Если , то и если , то переходим на **Шаг 2**. Если , то и если , то переходим на **Шаг 2**, иначе СТОП.

Не трудно заметить, что количество динамически неустойчивых решений будет меньше, чем общее количество запусков алгоритма. Из этого следует, что оценка , и чем выше её значение, тем выше уровень динамической устойчивости и тем лучше работает исследуемый алгоритм.

## Результаты проверки на динамическую устойчивость

Для проверки уровня динамической устойчивости муравьиного алгоритма для задачи CVRP были взяты 5 тестовых задач из стандартной библиотеки Capacitated Vehicle Routing Problem Library [13]. Значение параметров: . Результаты приведены в Таблица 1.

Для каждой из задач было сгенерировано по 25 различных решений. Для каждого решения проводилось 10 тестов. Исходный маршрут разбивался на 10 периодов. Если решение становилось в ходе проверки динамически неустойчивым, то текущий тест прекращался, а результат фиксировался. После завершения 10-го периода проводить эксперименты не имеет смысла, т.к. маршрут обхода городов будет закончен. Все полученные результаты по данному эксперименту представлены в [Приложение 2].

|  |  |  |  |
| --- | --- | --- | --- |
| Задача | Кол-во экспер. | Кол-во динамически неустойчивых решений | дин.уст. решения |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| E-n76-k8 | 250 | 163 | 47 | 15 | 20 | 5 | 0 | 0 | 0 | 0 | 0 |
| E-n101-k14 | 250 | 139 | 51 | 14 | 27 | 15 | 4 | 0 | 0 | 0 | 0 |
| M-n151-k12 | 250 | 171 | 39 | 24 | 11 | 0 | 1 | 1 | 3 | 0 | 0 |
| M-n200-k17 | 250 | 155 | 66 | 26 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| X-n256-k16 | 250 | 172 | 36 | 13 | 17 | 3 | 1 | 8 | 0 | 0 | 0 |
| Всего | 1250 | 800 | 239 | 92 | 78 | 23 | 6 | 9 | 3 | 0 | 0 |

Таблица . Оценка динамической устойчивости ACO для CVRP

В данном эксперименте не удалось сгенерировать динамически устойчивое решение. Это объясняется недостаточным количеством итераций муравьиного алгоритма. Большинство решений теряли устойчивость уже при первом периоде проверки динамической устойчивости, следовательно, при уменьшении размерности задачи удавалось выбраться из зоны локального минимума и найти решение лучше. Данный минус алгоритма исправляется использованием дополнительных алгоритмов локального поиска на полученных решениях в конце каждой итерации, таких как 2-opt и 3-opt [17].

Среднее значение экспериментального уровня динамической устойчивости муравьиного алгоритма для решения задачи Capacity Vehicle Routing Problem:

Данное значение является низким: все сгенерированные на начальном этапе решения не сохраняют свойство оптимальности в процессе своей реализации. Это означает, что существуют другие маршруты, которые можно получить динамически с помощью того же эвристического алгоритма, при этом значения их целевой функции будут меньше по сравнению с начальным решением. Алгоритм динамического улучшения эвристических алгоритмов будет подробнее рассмотрен в следующей главе.

# Динамическая адаптация

Используя определения динамической устойчивости и проанализировав полученные данные в предыдущей главе, заметим, что, если после периода начальное решение не соответствует критерию динамической устойчивости, т.е. для оставшейся части узлов было найдено решение лучше, то транспортному средству 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 [18], 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, обширно исследуется в области теории игр [16][19][20].

## Динамическая адаптация для ACO

Динамическая адаптация(DA) эвристического алгоритма заключается в следующем:

Генерируем решение с помощью алгоритма ACO для всех узлов. Это и будет начальным решением.

Маршрут разбивается на частей.

После прохождения каждой части для оставшихся узлов генерируем решение с помощью той же эвристики. Если получили 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 способствовать нахождению локальных или глобальных оптимумов. Блок-схема алгоритма DAACO изображена на Рис 2.



Рис 2. Блок схема алгоритма DAACO.

Реализации основных классов алгоритма представлены в [Приложение 5].

# Оценка вычислительной сложности алгоритма DAACO

Вычислительная сложность алгоритма муравьиной колонии, как представлено в статье [21], определяется формулой (13). Она зависит от количества итераций (линейная зависимость), количества муравьев в колонии (линейная зависимость) и количества клиентов , которых нужно обслужить (квадратичная зависимость). Процесс обновления феромонов представлен последним членом (квадратичная зависимость):

 (13)

Рассчитаем вычислительную сложность для динамической адаптации муравьиного алгоритма. Его основная задача разбивать решение на частей и, постепенно уменьшая размерность задачи, запускать эвристику. Тогда общий вид формулы вычислительной сложности динамической адаптации примет вид:

 (14)

Где – вычислительная сложность эвристического алгоритма для задачи размерности . Подставив в формулу (14) вычислительную сложность для муравьиной колонии, получим следующее:

Как можем заметить, вычислительная сложность динамической адаптации муравьиного алгоритма отличается от вычислительной сложности обычного муравьиного алгоритма на линейный множитель . Это приводит лишь к небольшому увеличению времени работы алгоритма, но благодаря своей особенности позволяет уменьшить значения параметров и что приведет к уменьшению значения прироста времени вычисления.

 Данные расчеты приводились для динамической адаптации с равномерным разбиением. Если разбиение производить по другому принципу, то, соответственно, изменится и линейный множитель.

# Анализ эффективности DAACO

Для анализа эффективности DAACO проведем некоторую серию тестов. Вначале оценим устойчивость решений, генерируемых динамической адаптацией муравьиного алгоритма. Затем проведем сравнение генерируемых решений по критерию точности. Третий тест основан на проверке точности генерации решений алгоритма DAACO за время работы алгоритма ACO для одной и той же задачи.

##  Оценка динамической устойчивости

Для оценки динамической устойчивости DAACO повторим эксперимент, который провели для алгоритма ACО. Сгенерируем решений для задач и проведем над ними по проверок динамической устойчивости с шагом (параметр разбиения для динамической адаптации будет такой же – . Все полученные результаты приведены в [Приложение 3]

|  |  |  |  |
| --- | --- | --- | --- |
| Задача | Кол-во экспер. | Кол-во динамически неустойчивых решений | дин.уст. Решения |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| E-n76-k8 | 250 | 1 | 2 | 2 | 8 | 5 | 30 | 14 | 17 | 0 | 171 |
| E-n101-k14 | 250 | 3 | 0 | 3 | 3 | 7 | 21 | 18 | 28 | 7 | 160 |
| M-n151-k12 | 250 | 0 | 1 | 1 | 0 | 1 | 4 | 34 | 56 | 20 | 133 |
| M-n200-k17 | 250 | 0 | 0 | 1 | 1 | 3 | 19 | 15 | 33 | 44 | 134 |
| X-n256-k16 | 250 | 0 | 3 | 2 | 3 | 5 | 16 | 20 | 36 | 49 | 116 |
| Всего | 1250 | 4 | 6 | 9 | 15 | 21 | 90 | 101 | 170 | 120 | 714 |

Таблица . Оценка динамической устойчивости DAACO для CVRP

Как можно заметить из Таблица 2, после применения динамической адаптации среднее значение экспериментального уровня динамической устойчивости стало отличным от 0 и равно 0,571

 Количество устойчивых решений уменьшается с увеличением размерности задач, это характеризуется тем, что для всех задач алгоритм запускался на одинаковых параметрах. Соответственно, увеличивая количество итераций и параметр разбиения для динамической адаптации относительно размерности задач, мы будем получать решения лучше, чем текущие, а среднее значение экспериментального уровня динамической устойчивости увеличится.

## Сравнение решений DAACO и ACO

Используя решения, полученные при проверке устойчивости решений ACO и DAACO, проведем сравнение лучших решений и средних значений.

|  |  |  |
| --- | --- | --- |
| задача | Среднее значение | Лучшее значение |
| ACO | DAACO | ACO | DAACO |
| E-n76-k8.vrp | 939,28 | 856,76 | 876 | 799 |
| E-n101-k14.vrp | 1412,24 | 1315,2 | 1368 | 1257 |
| M-n151-k12.vrp | 1413,64 | 1272,84 | 1357 | 1214 |
| M-n200-k17.vrp | 1863,08 | 1703,92 | 1810 | 1600 |
| X-n256-k16.vrp | 23241 | 21841,8 | 22666 | 21288 |

Таблица . Сравнение точности алгоритмов

Из данных в Таблица 3 видно, что среднее значение генерируемых решений алгоритмом DAACO меньше, чем алгоритмом ACO.

Вычислим процент улучшения по формуле:

Где — значение целевой функции решения полученного муравьиным алгоритмом для задачи , – значение целевой функции решения полученного динамической адаптацией муравьиного алгоритма.

 Результаты видны в Таблица 4:

|  |  |
| --- | --- |
| задача | коэффициент улучшения |
| Среднее значение | Максимальное значение |
| E-n76-k8.vrp | 8,79 | 8,79 |
| E-n101-k14.vrp | 6,87 | 8,11 |
| M-n151-k12.vrp | 9,96 | 10,54 |
| M-n200-k17.vrp | 8,54 | 11,60 |
| X-n256-k16.vrp | 6,02 | 6,08 |

Таблица . Улучшение точности в процентах

Средний процент улучшения решений составил примерно 8%.

## Эффективность DAACO за время работы ACO

Так как вычислительная сложность DAACO больше, чем у ACO, то, уменьшив значения параметров и для ACO, так, чтобы время работы DAACO и ACO сравнялось, мы сможем провести сравнение точности этих алгоритмов. Значение разбиения на периоды , тогда новые значения параметров должны удовлетворять неравенству (15):

 (15)

 Алгоритмы запускались для трёх задач по 5 раз. Для обычного муравьиного алгоритма использовался параметр , а для DAACO , количеству клиентов в задаче. В Таблица 5 приведено сравнение результатов средних значений целевой функции и времени работы алгоритмов, все результаты предоставлены в []

|  |  |  |
| --- | --- | --- |
| задача | ср.значение эффективности | ср. значение времени в с. |
| ACO | DAACO | ACO | DAACO |
| E-n76-k8.vrp | 924,2 | 859 | 63,10 | 55,64 |
| E-n101-k14.vrp | 1393,8 | 1302,8 | 137,41 | 119,70 |
| M-n151-k12.vrp | 1402,6 | 1298,4 | 467,19 | 435,60 |

Таблица . Сравнение средних показателей эффективности и времени

Из результатов видно, что динамическая адаптация муравьиного алгоритма вычисляет решение лучше, чем обычный муравьиный алгоритм, при этом за почти одинаковое время работы.

# Заключение

В работе была исследована одна из задач транспортной маршрутизации – CVRP, а так же один из классических эвристических алгоритмов для её решения – ACO.

Подробно изучен, описан и реализован на языке программирования Java муравьиный алгоритм (ACO) для решения задачи CVRP, приведены полученные экспериментальным путем наборы параметров для запуска муравьиного алгоритма. Произведена его оценка уровня динамической устойчивости решений, генерируемых данной эвристикой. Из-за того, что эвристики не обеспечивают получение оптимальных решений, но генерирует решения близкие к оптимальным, среднее значение этой величины для рассмотренных задач получилось равно 0, из чего делается вывод, что значение итераций для ACO было недостаточно высоким, но и оно не гарантирует отсутствие зацикливания алгоритма на локальном минимуме. Таким образом, использовать чистую эвристику ACO без алгоритмов локального поиска даже при больших значениях итераций и количества муравьиных агентов не целесообразно.

Принцип динамической устойчивости показал, что любой эвристический 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 0,571. Также значения целевых функций решений, сгенерированных алгоритмом DAACO, и их среднее меньше, чем полученные с помощью классического муравьиным алгоритмом. Средний процент улучшения решений равен 8%. К тому же, динамическая адаптация муравьиного алгоритма показала лучшие результаты на меньших значениях параметров итераций алгоритма, затратив на вычисление меньше времени, чем обычный муравьиный алгоритм.

Результаты экспериментов показали, что использование динамической адаптации муравьиного алгоритма будет эффективней, чем использование простого муравьиного алгоритма, точность вычисления увеличивается, уменьшается разброс получаемых решений, а главное, позволяет уменьшить время вычислений за счет уменьшения значений параметров итераций и численности муравейника, не ухудшив значения генерируемых решений и не используя посторонние способы улучшения.

# Список литературы

1. Dantzig G. B., Ramser J. H., The Truck Dispatching Problem, 1959.
2. [Braekersab](https://www.sciencedirect.com/science/article/abs/pii/S0360835215004775#!) K., [Ramaekersa](https://www.sciencedirect.com/science/article/abs/pii/S0360835215004775#!) K., [Van Nieuwenhuysec](https://www.sciencedirect.com/science/article/abs/pii/S0360835215004775#!) I. The vehicle routing problem: State of the art classification and review. [Computers & Industrial Engineering](https://www.sciencedirect.com/science/journal/03608352). [Volume 99](https://www.sciencedirect.com/science/journal/03608352/99/supp/C), September 2016, Pages 300-313.
3. Toth, P., Vigo, D.(2002). Models, relaxations and exact approaches for capacitated vehicle routing problem. Discrete Applied Mathematics, 123: 487-512.
4. Janacek, J., Janosikova, L., Kohani, M. (2013). Modelovanie a optimalizacia. EDIS vydavatelstvo ZU, in Slovak. (2013).
5. M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. Artificial Life, 5:137–172, 1999.
6. J.-L. Deneubourg, S. Aron, and S. Gossand J.-M. Pasteels. The self-organizing exploratory pattern of the Argentine ant. Journal of Insect Behavuiour, 3: 159–168, 1990.
7. M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1(1):53–66, 1997.
8. L.M. Gambardella and M. Dorigo. Ant-Q: a reinforcement learning approach to the travelling salesman problem. In Proceedings of the Twelfth International Conference on Machine Learning, ML-95. Morgan Kaufmann, Palo Alto, California, USA, 1995.
9. L.M. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In IEEE Conference on Evolutionary Computation (ICEC96), pages 622–627, 1996.
10. C.J. Watkins and P. Dayan. Q-learning. Machine Learning, 8:279–292, 1992.
11. B. Bullnheimer, R. F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In Proceedings of the 2nd International Conference on Metaheuristics - MIC97. INRA Sophia-Antipolis & PRiSM Versailles, 1997.
12. B. Bullnheimer, R.F. Hartl, and C. Struss. An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research, 89:319–328, 1999.
13. Capacitated Vehicle Routing Problem Library. <http://vrp.atd-lab.inf.puc-rio.br/index.php/en/>.
14. 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.
15. 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.
16. Петросян Леон Аганесович и Зенкевич Николай Анатольевич. Принципы устойчивой кооперации. Управление большими системами: сборник трудов, (3):100–120, 2009.
17. Dorigo, M., Stutzle, T. Ant colony optimization. The MIT press, 2004
18. 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.
19. V. Zakharov and M. Dementieva. Multistage cooperative games and problem of time consistency. International Game Theory Review, 6:157–170, 2004.
20. 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.
21. P. Stodola, J. Mazal, M. Podhorec, O. Litva. Using the Ant Colony Optimization Algorithm for the Capacitated Vehicle Routing. Proceedings of the 16th International Conference on Mechatronics - Mechatronika 2014

# Приложение 1

Лучшие комбинации параметров для ACO

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1 | 1,3 | 0,64 | 4 |
| 1 | 1,4 | 0,64 | 4 |
| 1,02 | 1,15 | 0,62 | 7 |
| 1,02 | 1,2 | 0,76 | 7 |
| 1,04 | 1,25 | 0,72 | 4 |
| 1,04 | 1,35 | 0,8 | 4 |
| 1,04 | 1,4 | 0,64 | 6 |
| 1,04 | 1,4 | 0,66 | 4 |
| 1,04 | 1,5 | 0,74 | 4 |
| 1,06 | 1 | 0,62 | 7 |
| 1,06 | 1 | 0,68 | 7 |
| 1,06 | 1,15 | 0,74 | 8 |
| 1,06 | 1,25 | 0,72 | 7 |
| 1,08 | 0,95 | 0,82 | 8 |
| 1,08 | 1 | 0,84 | 6 |
| 1,08 | 1 | 0,9 | 7 |
| 1,08 | 1,05 | 0,74 | 7 |
| 1,08 | 1,05 | 0,76 | 5 |
| 1,08 | 1,05 | 0,88 | 7 |
| 1,08 | 1,1 | 0,88 | 6 |
| 1,08 | 1,15 | 0,86 | 5 |
| 1,08 | 1,2 | 0,64 | 5 |
| 1,08 | 1,2 | 0,68 | 7 |
| 1,08 | 1,25 | 0,88 | 7 |
| 1,08 | 1,3 | 0,62 | 8 |
| 1,08 | 1,4 | 0,84 | 7 |
| 1,1 | 0,95 | 0,9 | 4 |
| 1,1 | 1,1 | 0,6 | 7 |
| 1,1 | 1,1 | 0,76 | 6 |
| 1,1 | 1,15 | 0,8 | 4 |
| 1,1 | 1,2 | 0,76 | 4 |
| 1,1 | 1,2 | 0,78 | 8 |
| 1,1 | 1,25 | 0,76 | 6 |
| 1,1 | 1,25 | 0,84 | 5 |
| 1,1 | 1,25 | 0,88 | 4 |
| 1,1 | 1,35 | 0,64 | 5 |

# Приложение 2

Результаты проверок на устойчивость решений алгоритма ACO

|  |  |  |
| --- | --- | --- |
| # | Solve | период деления |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 923 | 3 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 951 | 4 | 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 950 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 915 | 1 | 5 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| 4 | 916 | 0 | 0 | 3 | 7 | 0 | 0 | 0 | 0 | 0 |
| 5 | 931 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 957 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 956 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 962 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 946 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 946 | 8 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 926 | 4 | 1 | 3 | 2 | 0 | 0 | 0 | 0 | 0 |
| 12 | 876 | 0 | 2 | 1 | 5 | 2 | 0 | 0 | 0 | 0 |
| 13 | 925 | 2 | 0 | 3 | 2 | 3 | 0 | 0 | 0 | 0 |
| 14 | 933 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 941 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 941 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 930 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 918 | 8 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 963 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 969 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 946 | 9 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 963 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | 950 | 4 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 948 | 7 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

Таблица 6. Задача E-n76-k8.vrp

|  |  |  |
| --- | --- | --- |
| # | Solve | Период деления |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1416 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1416 | 6 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1420 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1433 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1435 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1422 | 6 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1405 | 2 | 7 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1408 | 2 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1423 | 4 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1399 | 9 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1389 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 1418 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 1380 | 0 | 7 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 1411 | 0 | 0 | 2 | 4 | 3 | 1 | 0 | 0 | 0 |
| 14 | 1444 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 1414 | 3 | 2 | 3 | 1 | 1 | 0 | 0 | 0 | 0 |
| 16 | 1434 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 1421 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 1373 | 0 | 1 | 0 | 5 | 3 | 1 | 0 | 0 | 0 |
| 19 | 1368 | 0 | 0 | 0 | 4 | 6 | 0 | 0 | 0 | 0 |
| 20 | 1424 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 1459 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 1399 | 0 | 1 | 0 | 6 | 1 | 2 | 0 | 0 | 0 |
| 23 | 1410 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 1385 | 2 | 0 | 0 | 7 | 1 | 0 | 0 | 0 | 0 |

Таблица 7. Задача E-n101-k14.vrp

|  |  |  |
| --- | --- | --- |
| # | Solve | Период деления |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1424 | 7 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1433 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1431 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1438 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1422 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1432 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1408 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1357 | 1 | 0 | 5 | 0 | 0 | 0 | 1 | 3 | 0 |
| 8 | 1393 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1419 | 4 | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1385 | 4 | 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 1435 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 1440 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 1383 | 2 | 3 | 3 | 2 | 0 | 0 | 0 | 0 | 0 |
| 14 | 1431 | 5 | 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 15 | 1413 | 9 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 1397 | 3 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 1408 | 6 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 1414 | 7 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 1406 | 7 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 1442 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 1447 | 9 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 1370 | 1 | 0 | 0 | 9 | 0 | 0 | 0 | 0 | 0 |
| 23 | 1419 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 1394 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Таблица . Задача M-n151-k12.vrp

|  |  |  |
| --- | --- | --- |
| # | Solve | Период деления |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1877 | 8 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1856 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1860 | 0 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1860 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1810 | 1 | 3 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1857 | 5 | 3 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1856 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1906 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1818 | 0 | 4 | 5 | 1 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1880 | 3 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1876 | 3 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 1847 | 3 | 6 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 1877 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 1846 | 7 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 1843 | 3 | 6 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 1889 | 8 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 16 | 1852 | 2 | 5 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 1889 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 1854 | 7 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 1839 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 1850 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 1851 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 1885 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | 1929 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 1870 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Таблица . Задача M-n200-k17.vrp

|  |  |  |
| --- | --- | --- |
| # | Solve | Период деления |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 23543 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 23446 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 22674 | 1 | 0 | 3 | 5 | 1 | 0 | 0 | 0 | 0 |
| 3 | 23036 | 2 | 2 | 1 | 5 | 0 | 0 | 0 | 0 | 0 |
| 4 | 23399 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 23324 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 23189 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 23340 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 23281 | 7 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 23090 | 0 | 2 | 4 | 3 | 1 | 0 | 0 | 0 | 0 |
| 10 | 23253 | 4 | 5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 11 | 23544 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 23307 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 23284 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 23090 | 3 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 22666 | 0 | 1 | 1 | 0 | 0 | 0 | 8 | 0 | 0 |
| 16 | 23083 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 23294 | 5 | 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 23636 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 23437 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 23000 | 2 | 3 | 0 | 4 | 0 | 1 | 0 | 0 | 0 |
| 21 | 23154 | 1 | 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 23326 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | 23273 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 23357 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Таблица . Задача X-n256-k16.vrp

# Приложение 3

Результаты проверок на устойчивость решений алгоритма DAACO

|  |  |  |
| --- | --- | --- |
| # | Solve | Период деления |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 876 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 852 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 844 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 830 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 907 | 1 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 5 | 838 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| 6 | 872 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 |
| 7 | 836 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 |
| 8 | 866 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 0 |
| 9 | 800 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 846 | 0 | 0 | 0 | 0 | 0 | 1 | 5 | 0 | 0 |
| 11 | 835 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 0 | 0 |
| 12 | 876 | 0 | 0 | 0 | 0 | 0 | 1 | 7 | 2 | 0 |
| 13 | 873 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 855 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 862 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5 | 0 |
| 16 | 873 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 17 | 877 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 18 | 824 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 907 | 0 | 0 | 1 | 2 | 3 | 4 | 0 | 0 | 0 |
| 20 | 849 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 |
| 21 | 843 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 22 | 909 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | 870 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 |
| 24 | 799 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Таблица . Задача E-n76-k8.vrp

|  |  |  |
| --- | --- | --- |
| # | Solve | Период деления |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1316 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1257 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1305 | 0 | 0 | 0 | 0 | 0 | 7 | 2 | 0 | 0 |
| 3 | 1316 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 4 | 1259 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| 5 | 1302 | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 1 | 0 |
| 6 | 1260 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1302 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| 8 | 1290 | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 0 | 0 |
| 9 | 1345 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 3 |
| 10 | 1282 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 1315 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 12 | 1325 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 13 | 1302 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 1315 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 9 | 0 |
| 15 | 1298 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 1372 | 0 | 0 | 0 | 1 | 2 | 5 | 2 | 0 | 0 |
| 17 | 1342 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 |
| 18 | 1323 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 6 | 0 |
| 19 | 1357 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 1 | 0 |
| 20 | 1308 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 1349 | 0 | 0 | 0 | 0 | 3 | 0 | 2 | 3 | 0 |
| 22 | 1322 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 23 | 1347 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 1371 | 2 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 |

Таблица . Задача E-n101-k14.vrp

|  |  |  |
| --- | --- | --- |
| # | Solve | Период деления |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1237 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 |
| 1 | 1252 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |
| 2 | 1307 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 6 | 0 |
| 3 | 1243 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1302 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1260 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 6 | 1278 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1292 | 0 | 0 | 0 | 0 | 1 | 2 | 5 | 0 | 0 |
| 8 | 1322 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| 9 | 1266 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 0 |
| 10 | 1216 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 0 |
| 11 | 1282 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 |
| 12 | 1249 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 4 |
| 13 | 1240 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 |
| 14 | 1314 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 6 | 0 |
| 15 | 1286 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 |
| 16 | 1299 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| 17 | 1257 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 4 | 0 |
| 18 | 1312 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 19 | 1307 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 |
| 20 | 1214 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| 21 | 1269 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 0 |
| 22 | 1294 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 7 | 0 |
| 23 | 1254 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 1269 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 |

Таблица . Задача M-n151-k12.vrp

|  |  |  |
| --- | --- | --- |
| # | Solve | период разбиения |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1728 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 6 |
| 1 | 1654 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 2 | 1652 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |
| 3 | 1722 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 3 |
| 4 | 1600 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 5 | 1669 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 1 | 0 |
| 6 | 1723 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 5 | 0 |
| 7 | 1715 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 8 | 1743 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 9 | 1652 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1737 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| 11 | 1682 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 1758 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 13 | 1774 | 0 | 0 | 0 | 0 | 1 | 5 | 0 | 2 | 0 |
| 14 | 1747 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 4 | 0 |
| 15 | 1685 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 16 | 1725 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 5 |
| 17 | 1711 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 |
| 18 | 1678 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 1640 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 3 |
| 20 | 1708 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 | 0 |
| 21 | 1747 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 |
| 22 | 1753 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 5 |
| 23 | 1698 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 24 | 1697 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 1 | 4 |

Таблица . Задача M-n200-k17.vrp

|  |  |  |
| --- | --- | --- |
| # | Solve | период деления |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 21987 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 2 | 2 |
| 1 | 22033 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 4 |
| 2 | 22137 | 0 | 1 | 0 | 0 | 0 | 3 | 2 | 0 | 3 |
| 3 | 22201 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 2 | 5 |
| 4 | 22025 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 1 | 2 |
| 5 | 22000 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 2 |
| 6 | 21585 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 7 | 21641 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 8 | 21700 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 |
| 9 | 22374 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 5 |
| 10 | 21837 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 4 | 2 |
| 11 | 21288 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 21382 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 22218 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 2 | 3 |
| 14 | 21819 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 3 |
| 15 | 22103 | 0 | 1 | 0 | 0 | 0 | 0 | 4 | 3 | 0 |
| 16 | 21842 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 4 |
| 17 | 21616 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |
| 18 | 21445 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 22181 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 20 | 21475 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 |
| 21 | 21438 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 21913 | 0 | 0 | 1 | 0 | 1 | 2 | 2 | 1 | 3 |
| 23 | 21775 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 0 |
| 24 | 22030 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 5 | 3 |

Таблица . Задача X-n256-k16.vrp

# Приложение 4

Здесь приведены данные времени работы алгоритма и полученное для этого решения значение целевой функции.

|  |  |  |  |
| --- | --- | --- | --- |
| № решения | E-n76-k8.vrp | E-n101-k14.vrp | M-n151-k12.vrp |
| J | время (с) | J | время (с) | J | время (с) |
| 0 | 938 | 62,22 | 1387 | 137,45 | 1416 | 480,92 |
| 1 | 916 | 63,89 | 1383 | 137,40 | 1406 | 479,34 |
| 2 | 909 | 64,24 | 1384 | 136,52 | 1422 | 457,14 |
| 3 | 928 | 63,29 | 1409 | 138,57 | 1383 | 448,96 |
| 4 | 930 | 61,86 | 1406 | 137,10 | 1386 | 469,58 |
| ср,знач | 924,2 | 63,10 | 1393,8 | 137,41 | 1402,6 | 467,19 |

Таблица . Результаты запуска алгоритма ACO

|  |  |  |  |
| --- | --- | --- | --- |
| № решения | E-n76-k8.vrp | E-n101-k14.vrp | M-n151-k12.vrp |
| J | время (с) | J | время (с) | J | время (с) |
| 0 | 886 | 52,72 | 1311 | 126,05 | 1293 | 417,31 |
| 1 | 850 | 56,79 | 1265 | 116,35 | 1248 | 466,57 |
| 2 | 807 | 55,85 | 1319 | 115,22 | 1257 | 460,19 |
| 3 | 910 | 58,48 | 1328 | 119,98 | 1362 | 403,59 |
| 4 | 842 | 54,35 | 1291 | 120,92 | 1332 | 430,34 |
| ср,знач | 859 | 55,64 | 1302,8 | 119,70 | 1298,4 | 435,60 |

Таблица . Результаты запуска алгоритма DAACO

# Приложение 5

Класс описывающий поведение муравья в ACO

**public abstract class** Ant **extends** Observable **implements** Runnable {
 **protected int numAntID**;
 **protected int curCapac**;
 **protected int curNode**;
 **protected int startNode**;
 **protected long pathValue**;
 **protected** Observer **observer**;
 **protected** Vector **pathVect**;
 **protected int intCounter**;
 **private static int** *numAntIDCounter* = 0;
 **private static** PrintStream *outs*;
 **protected static** AntColony *antColony*;
 **public static long** *bestPathValue* = Long.***MAX\_VALUE***;
 **public static** Vector *bestPathVect* = **null**;
 **public static int** *lastBestPathIteration* = 0;
 **public static int** *maxCap*;
 **public static void** setAntColony(AntColony antColony) {
 Ant.*antColony* = antColony;
 }
 **public static void** reset() {
 *bestPathValue* = Long.***MAX\_VALUE***;
 *bestPathVect* = **null**;
 *lastBestPathIteration* = 0;
 *outs* = **null**;
 }
 **public** Ant(**int** nStartNode, Observer observer, **int** cap) {
 *numAntIDCounter*++;
 **numAntID** = *numAntIDCounter*;
 **startNode** = nStartNode;
 **this**.**observer** = observer;
 *maxCap* = cap;
 **intCounter** = 0;
 }
 **public void** init(PartPath partPath) {
 **if** (*outs* == **null**) {
 **try** {
 *outs* = **new** PrintStream(**new** FileOutputStream(TestCVRP.*folderName* + **"\\"** + *antColony*.getID() + **"\_"** + *antColony*.getGraph().nodes() + **"x"** + *antColony*.getAnts() + **"x"** + *antColony*.getIterations() + **"\_ants.txt"**));
 } **catch** (Exception ex) {
 ex.printStackTrace();
 }
 }
 **final** AntGraph graph = *antColony*.getGraph();
 **curNode** = partPath.getCurNode();
 **pathVect** = **new** Vector(graph.nodes());
 **pathVect**.addAll(partPath.getPartPathVect());
 **pathValue** = partPath.getPartPathValue();
 **curCapac** = partPath.getCurCup();
 }
 **public void** start(PartPath partPath) {
 init(partPath);
 Thread thread = **new** Thread(**this**);
 thread.setName(**"Ant "** + **numAntID**);
 thread.start();
 }
 **public void** run() {
 **intCounter**++;
 **final** AntGraph graph = *antColony*.getGraph();
 **while** (!end()) {
 **int** nNewNode;
 **synchronized** (graph) {
 nNewNode = stateTransitionRule(**curNode**);
 **pathValue** += graph.delta(**curNode**, nNewNode);
 }
 **pathVect**.addElement(**new** Integer(nNewNode));
 **curNode** = nNewNode;
 }
 **synchronized** (graph) {
 localUpdatingRule(**pathVect**, **pathValue**);
 }
 **synchronized** (graph) {
 **if** (better(**pathValue**, *bestPathValue*)) {
 *bestPathValue* = **pathValue**;
 *bestPathVect* = **pathVect**;
 *lastBestPathIteration* = **intCounter**;
 *outs*.println(**"Ant "** + **numAntID** + **", Лучшая длина "** + *bestPathValue* + **", Итерация "** + *lastBestPathIteration* + **", длина "** + *bestPathVect*.size() + **", маршрут"** + *bestPathVect*);
 }
 }
 **observer**.update(**this**, **null**);
 **if** (*antColony*.done())
 *outs*.close();
 }
 **protected abstract boolean** better(**double** dPathValue, **double** dBestPathValue);
 **public abstract int** stateTransitionRule(**int** r);
 **public abstract void** localUpdatingRule(Vector path, **double** length);
 **public abstract boolean** end();
 **public static int**[] getBestPath() {
 **int** nBestPathArray[] = **new int**[*bestPathVect*.size()];
 **for** (**int** i = 0; i < *bestPathVect*.size(); i++) {
 nBestPathArray[i] = ((Integer) *bestPathVect*.elementAt(i)).intValue();
 }
 **return** nBestPathArray;
 }
 **public** String toString() {
 **return "Ant "** + **numAntID** + **":"** + **curNode**;
 }
}

Класс описывающий поведение муравьиной колонии

**public abstract class** AntColony **implements** Observer {
 **protected** PrintStream **outs**;
 **protected** AntGraph **graph**;
 **protected** Ant[] **ants**;
 **protected int numAnts**;
 **protected int numAntCounter**;
 **protected int iterCounter**;
 **protected int numIterat**;
 **protected int capacity**;
 **private int id**;
 **private static int** *iDCounter* = 0;
 **public** AntColony(AntGraph graph, **int** nAnts, **int** nIterations, **int** capacity) {
 **this**.**graph** = graph;
 **numAnts** = nAnts;
 **numIterat** = nIterations;
 *iDCounter*++;
 **id** = *iDCounter*;
 **this**.**capacity** = capacity;
 **iterCounter** = 0;
 }
 **public synchronized void** start() {
 **ants** = createAnts(**graph**, **numAnts**);
 **try** {
 **outs** = **new** PrintStream(**new** FileOutputStream(TestCVRP.*folderName* + **"\\"** + **id** + **"\_"** + **graph**.nodes() + **"x"** + **ants**.**length** + **"x"** + **numIterat** + **"\_colony.txt"**));
 } **catch** (Exception ex) {
 ex.printStackTrace();
 }
 PartPath dynamicAdaptaion = **new** PartPath(**graph**.nodes(), **capacity**);
 **while** (dynamicAdaptaion.exitCondition()) {
**iterCounter** = 0;
 **graph**.resetTau();
 **while** (**iterCounter** < **numIterat**) {
 *// run an iteration* iteration(dynamicAdaptaion);
 **try** {
 wait();
 } **catch** (InterruptedException ex) {
 ex.printStackTrace();
 }
 **synchronized** (**graph**) {
 globalUpdatingRule();
 }
 }
 dynamicAdaptaion.routeDivision(*bestPathVect*, *bestPathValue*, **graph**);}
 **if** (**iterCounter** == **numIterat**) {
 **outs**.close();
 }
 }
 **private void** iteration(PartPath partPath) {
 **numAntCounter** = 0;
 **iterCounter**++;
 **outs**.print(**"Итерация "** + **iterCounter**);
 **for** (**int** i = 0; i < **ants**.**length**; i++) {
 **ants**[i].start(partPath);
 }
 }
 **public** AntGraph getGraph() {
 **return graph**;
 }
 **public int** getAnts() {
 **return ants**.**length**;
 }
 **public int** getIterations() {
 **return numIterat**;
 }
 **public int** getID() {
 **return id**;
 }
 **public synchronized void** update(Observable ant, Object obj) {
 **outs**.print(**"; "** + ((Ant) ant).**pathValue**);
 **numAntCounter**++;
 **if** (**numAntCounter** == **ants**.**length**) {
 **outs**.println(**"; iteration: "** + Ant.*lastBestPathIteration* + **"; result: "** + Ant.*bestPathValue*);
 System.***out***.println(**"---------------------------"**);
 System.***out***.println(**iterCounter** + **" - Best Path: "** + Ant.*bestPathValue*);
 System.***out***.println(**"---------------------------"**);
 System.***out***.println(**"Path seq: "**);
 **for** (**int** i = 0; i < Ant.*bestPathVect*.size(); i++) {
 System.***out***.print(Ant.*bestPathVect*.get(i));
 System.***out***.print(**" "**);
 }
 System.***out***.println(**"\n"**);
 notify();
 }
 }
 **public double** getBestPathValue() {
 **return** Ant.*bestPathValue*;
 }
 **public int** getLastBestPathIteration() {
 **return** *lastBestPathIteration*;
 }
 **public boolean** done() {
 **return iterCounter** == **numIterat**;
 }
 **protected abstract** Ant[] createAnts(AntGraph graph, **int** ants);
 **protected abstract void** globalUpdatingRule();
}

Класс описывающий перемещение по графу

**public class** AntGraph **implements** Serializable {
 **private double**[][] **delta**;
 **private double**[][] **tau**;
 **private int**[] **demand**;
 **private int numNodes**;
 **private double tau0**;
 **public** AntGraph(**int** nNodes, **double**[][] delta, **double**[][] tau, **int**[] demand) {
 **if** (delta.**length** != nNodes)
 **throw new** IllegalArgumentException();
 **if** (demand.**length** != nNodes)
 **throw new** IllegalArgumentException();
 **numNodes** = nNodes;
 **this**.**delta** = delta;
 **this**.**tau** = tau;
 **this**.**demand** = demand;
 }
 **public** AntGraph(**int** nodes, **double**[][] delta, **int**[] demand) {
 **this**(nodes, delta, **new double**[nodes][nodes], demand);
 resetTau();
 }
 **public synchronized double** delta(**int** r, **int** s) {
 **return delta**[r][s];
 }
 **public synchronized double** tau(**int** r, **int** s) {
 **return tau**[r][s];
 }
 **public synchronized double** etha(**int** r, **int** s) {
 **return** ((**double**) 1) / delta(r, s);
 }
 **public synchronized int** nodes() {
 **return numNodes**;
 }
 **public synchronized int** demand(**int** node) {
 **return demand**[node];
 }
 **public synchronized void** updateTau(**int** r, **int** s, **double** value) {
 **tau**[r][s] = value;
 }
 **public void** resetTau() {
 **double** dAverage = averageDelta();
 **tau0** = (**double**) 1 / ((**double**) **numNodes** \* (0.5 \* dAverage));
 System.***out***.println(**"Average: "** + dAverage);
 System.***out***.println(**"Tau0: "** + **tau0**);
 **for** (**int** r = 0; r < nodes(); r++) {
 **for** (**int** s = 0; s < nodes(); s++) {
 **tau**[r][s] = **tau0**;
 }
 }
 }
 **public double** averageDelta() {
 **return** average(**delta**);
 }
 **public** String toString() {
 String str = **""**;
 String str1 = **""**;
 **for** (**int** r = 0; r < nodes(); r++) {
 **for** (**int** s = 0; s < nodes(); s++) {
 str += delta(r, s) + **"\t"**;
 str1 += tau(r, s) + **"\t"**;
 }
 str += **"\n"**;
 }
 **return** str + **"\n\n\n"** + str1;
 }
 **private double** average(**double** matrix[][]) {
 **double** dSum = 0;
 **for** (**int** r = 0; r < **numNodes**; r++) {
 **for** (**int** s = 0; s < **numNodes**; s++) {
 dSum += matrix[r][s];
 }
 }
 **double** dAverage = dSum / (**double**) (**numNodes** \* **numNodes**);
 **return** dAverage;
 }
}

Реализация поведения муравья для задачи CVRP

**public class** AntCVRP **extends** Ant {
 **private static final double *A*** = 1.04;
 **private static final double *B*** = 1.5;
 **public static final double *U*** = 4;
 **public static final double *P*** = 0.74;
 **private** AntGraph **graph**;
 **protected** Hashtable **m\_nodesToVisitTbl**;
 **public** AntCVRP(**int** nStartNode, Observer observer, **int** cap) {
 **super**(nStartNode, observer, cap);
 }
 **public void** init(PartPath partPath) {
 **super**.init(partPath);
 **graph** = *antColony*.getGraph();
 **m\_nodesToVisitTbl** = **new** Hashtable(partPath.getNodesToVisitTbl());
 }
 @Override
 **public int** stateTransitionRule(**int** r) {
 **return** chooseNext();
 }
 **private int** chooseNext() {
 **int** nMaxNode = 0;
 **double** dSum = 0;
 **int** nNode;
 Hashtable nodesToPosibleVisitTbl = **new** Hashtable();
 Enumeration en = **m\_nodesToVisitTbl**.elements();
 **while** (en.hasMoreElements()) {
 nNode = ((**int**) en.nextElement());
 **if** (**graph**.demand(nNode) <= **curCapac**) {
 nodesToPosibleVisitTbl.put(nNode, nNode);
 dSum += hValue(nNode);
 }
 }
 **if** (dSum != 0) {
 *// генерация выбора маршрута* **double** rand = Math.*random*();
 **double** segment = 0;
 en = nodesToPosibleVisitTbl.elements();
 **while** (en.hasMoreElements()) {
 nNode = (Integer) en.nextElement();
 segment += hValue(nNode) / dSum;
 **if** (rand < segment) {
 nMaxNode = nNode;
 **break**;
 }
 }
 }
 **if** (nMaxNode != 0) {
 **m\_nodesToVisitTbl**.remove(**new** Integer(nMaxNode));
 **curCapac** -= **graph**.demand(nMaxNode);
 } **else** {
 **curCapac** = *maxCap*;
 }
 **return** nMaxNode;
 }
 **private double** hValue(**int** nNode) {
 **double** value = Math.*pow*(**graph**.tau(**curNode**, nNode), ***A***) \* Math.*pow*(**graph**.etha(**curNode**, nNode), ***B***);
 **return** value == 0 ? Double.***MIN\_VALUE*** : value;
 }
 @Override
 **public void** localUpdatingRule(Vector path, **double** length) {
 **final** AntGraph graph = *antColony*.getGraph();
 **for** (**int** i = 1; i < path.size(); i++) {
 **int** currVertex = (**int**) path.get(i - 1);
 **int** nextVertex = (**int**) path.get(i);
 **double** val = graph.tau(currVertex, nextVertex) + ***U*** / (***P*** \* length);
 graph.updateTau(currVertex, nextVertex, val);
 **double** val2 = graph.tau( nextVertex,currVertex) + ***U*** / (***P*** \* length);
 graph.updateTau(nextVertex, currVertex , val2);
 }
 }
 @Override
 **public boolean** better(**double** dPathValue1, **double** dPathValue2) {
 **return** dPathValue1 < dPathValue2;
 }
 @Override
 **public boolean** end() {
 **return m\_nodesToVisitTbl**.isEmpty() && **curNode** == **startNode**;
 }
}

Реализация муравьиной колонии для задачи CVRP

**public class** AntCVRP **extends** Ant {
 **private static final double *A*** = 1.04;
 **private static final double *B*** = 1.5;
 **public static final double *U*** = 4;
 **public static final double *P*** = 0.74;
 **private** AntGraph **graph**;
 **protected** Hashtable **m\_nodesToVisitTbl**;
 **public** AntCVRP(**int** nStartNode, Observer observer, **int** cap) {
 **super**(nStartNode, observer, cap);
 }
 **public void** init(PartPath partPath) {
 **super**.init(partPath);
 **graph** = *antColony*.getGraph();
 **m\_nodesToVisitTbl** = **new** Hashtable(partPath.getNodesToVisitTbl());
 }
 @Override
 **public int** stateTransitionRule(**int** r) {
 **return** chooseNext();
 }
 **private int** chooseNext() {
 **int** nMaxNode = 0;
 **double** dSum = 0;
 **int** nNode;
 Hashtable nodesToPosibleVisitTbl = **new** Hashtable();
 Enumeration en = **m\_nodesToVisitTbl**.elements();
 **while** (en.hasMoreElements()) {
 nNode = ((**int**) en.nextElement());
 **if** (**graph**.demand(nNode) <= **curCapac**) {
 nodesToPosibleVisitTbl.put(nNode, nNode);
 dSum += hValue(nNode);
 }
 }
 **if** (dSum != 0) {
 *// генерация выбора маршрута* **double** rand = Math.*random*();
 **double** segment = 0;
 en = nodesToPosibleVisitTbl.elements();
 **while** (en.hasMoreElements()) {
 nNode = (Integer) en.nextElement();
 segment += hValue(nNode) / dSum;
 **if** (rand < segment) {
 nMaxNode = nNode;
 **break**;
 }
 }
 }
 **if** (nMaxNode != 0) {
 **m\_nodesToVisitTbl**.remove(**new** Integer(nMaxNode));
 **curCapac** -= **graph**.demand(nMaxNode);
 } **else** {
 **curCapac** = *maxCap*;
 }
 **return** nMaxNode;
 }
 **private double** hValue(**int** nNode) {
 **double** value = Math.*pow*(**graph**.tau(**curNode**, nNode), ***A***) \* Math.*pow*(**graph**.etha(**curNode**, nNode), ***B***);
 **return** value == 0 ? Double.***MIN\_VALUE*** : value;
 }
 @Override
 **public void** localUpdatingRule(Vector path, **double** length) {
 **final** AntGraph graph = *antColony*.getGraph();
 **for** (**int** i = 1; i < path.size(); i++) {
 **int** currVertex = (**int**) path.get(i - 1);
 **int** nextVertex = (**int**) path.get(i);
 **double** val = graph.tau(currVertex, nextVertex) + ***U*** / (***P*** \* length);
 graph.updateTau(currVertex, nextVertex, val);
 **double** val2 = graph.tau( nextVertex,currVertex) + ***U*** / (***P*** \* length);
 graph.updateTau(nextVertex, currVertex , val2);
 }
 }
 @Override
 **public boolean** better(**double** dPathValue1, **double** dPathValue2) {
 **return** dPathValue1 < dPathValue2;
 }
 @Override
 **public boolean** end() {
 **return m\_nodesToVisitTbl**.isEmpty() && **curNode** == **startNode**;
 }
}

Класс проводящий динамическую адаптацию алгоритма ACO

**public class** PartPath {
 **private long fullPathValue** = Long.***MAX\_VALUE***;
 **private** Vector **fullPathVect** = **null**;
 **private** Hashtable **nodesToVisitTbl**;
 **private long partPathValue**;
 **private** Vector **partPathVect**;
 **private int curNode**;
 **private int posintionOnPath**;
 **private int curCup**;
 **private int maxCup**;
 **private static int** *delta*;
 **private static int** *part*;
 **private static int** *iterate*;
 **public** PartPath(**int** nodes, **int** capacity) {
 *part* = 0;
 *iterate* = 0;
 **partPathVect** = **new** Vector();
 **posintionOnPath** = 1;
 **partPathValue** = 0;
 **curNode** = 0;
 **maxCup** = capacity;
 **curCup** = capacity;
 **nodesToVisitTbl** = **new** Hashtable();
 **for** (**int** i = 0; i < nodes; i++)
 **nodesToVisitTbl**.put(i, i);
 **partPathVect**.add(0);
 **nodesToVisitTbl**.remove(0);
 }
 **public void** routeDivision(Vector solvePathVect, **long** solvePathValue, AntGraph graph) {
 **if** (solvePathValue < **fullPathValue**) {
 **fullPathValue** = solvePathValue;
 **fullPathVect** = solvePathVect;
 }
 *ricePart*();
 **if** (*part* >= *delta*) {
 **return**;
 }
 **double** measure = **fullPathVect**.size() \* *part* \* 1.0 / *delta*;
 **while** (**posintionOnPath** < measure) {
 **int** newNode = (**int**) **fullPathVect**.get(**posintionOnPath**);
 **partPathVect**.addElement(newNode);
 **partPathValue** += graph.delta(**curNode**, newNode);
 **if** (newNode == 0) {
 **curCup** = **maxCup**;
 } **else** {
 **curCup** -= graph.demand(newNode);
 **nodesToVisitTbl**.remove(newNode);
 }
 **curNode** = newNode;
 **posintionOnPath**++;
 }
 }
 **public boolean** exitCondition() {
 **return** *part* < *delta*;
 }
 **public static void** setDelta(**int** delta) {
 PartPath.*delta* = delta;
 }
 **static void** ricePart() {
 *iterate*++;
 *part* = *iterate*; *//равномерное деление
 // part = (1 + iterate) \* iterate / 2; // по арифмитической прогррессии* }
 **public** Hashtable getNodesToVisitTbl() {
 **return nodesToVisitTbl**;
 }
 **public long** getPartPathValue() {
 **return partPathValue**;
 }
 **public** Vector getPartPathVect() {
 **return partPathVect**;
 }
 **public int** getCurNode() {
 **return curNode**;
 }
 **public int** getCurCup() {
 **return curCup**;
 }
}