Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра информатики и программирования

РЕАЛИЗАЦИЯ И СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ РАЗЛИЧНЫХ ТИПОВ ДЛЯ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЕРА

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студентки 4 курса 441 группы
направления 02.03.03 Математическое обеспечение и администрирование информационных систем
факультета компьютерных наук и информационных технологий
Крыловой Инессы Валерьевны

Научный руководитель:
Доцент, к.ф.-м.н. _____ М. В. Огнева
подпись, дата

Зав. кафедрой:
Доцент, к.ф.-м.н. _____ А.Г. Федорова
подпись, дата

Саратов 2016

Введение. Задача коммивояжера – одна из самых популярных и трудных задач комбинаторной оптимизации. Она заключается в поиске самого выгодного маршрута, проходящего через города только по одному разу с возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, и тому подобное).

Задача относится к числу трансвычислительных: уже при относительно небольшом числе городов (от 60) она не может быть решена методом перебора вариантов никакими компьютерами за время, меньшее нескольких миллиардов лет. Благодаря попыткам решить эту проблему, существует множество методов определения выгодного маршрута на разумном количестве вершин. Эта задача является технически интересной, и она широко распространена и актуальна в повседневной жизни. Например, проектирование сети электроснабжения между населенными пунктами, расстановка технических объектов на большой территории, облет экспедиций и сброс грузов, и т.д.. Оптимальное решение задачи минимизирует затраты на поездки, поэтому важно исследовать самые мощные инструменты ее решения.

Целью работы является реализация и сопряженный с ней сравнительный анализ методов решения задачи коммивояжера. Задачи в рамках этой цели:

1. Описать формулировку задачи коммивояжера.
2. Рассмотреть теоретически методы решения ЗКВ: полный перебор, метод ветвей и границ, жадный алгоритм, генетический алгоритм, муравьиный алгоритм и динамическое программирование.
3. Написать программы реализации 6 рассмотренных методов, протестировать их на различных входных данных.
4. Провести сравнительный анализ результатов по определенным параметрам.
5. Изучить, как распараллеливание повлияло на работу программ и их показатели эффективности.

Основная часть. *Задача коммивояжёра* – задача комбинаторной оптимизации, заключающаяся в отыскании самого выгодного маршрута,

проходящего через указанные города ровно по одному разу с последующим возвратом в исходный город. В таком случае выбор осуществляется среди гамильтоновых циклов. Это важная задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок. Задача при небольшом числе вершин может быть решена перебором всех вариантов. Но количество возможных маршрутов очень быстро возрастает – оно равно n!. И мощная ЭВМ, способная перебирать миллион вариантов в секунду, будет биться с задачей для 100 городов на протяжении 310144 лет. Поэтому задача относится к классу NP-полных задач: они практически неразрешимы, и неизвестны алгоритмы, способные решить их за разумное время. Ранним вариантом задачи может считаться игра Icosian Game Уильяма Гамильтона 19 века – чтобы найти маршруты на графе с 20 узлами.

Обоснование выбора методов. Методы были взяты для рассмотрения, так как достаточно популярны, интуитивно понятны и наглядны. Входными графами являются связные графы с циклами. Каждый граф представлен матрицей весов, на ее основе создается список ребер или смежности.

Метод полного перебора. В реализации перебора возможно применение 2 условий, сокращающих время и отсекающих неоптимальные пути:

1. Если длина нового пути превышает найденный локальный минимум – пропускать этот вариант и все, которые он порождает.

2. Если выбранное ребро пересекает одно из ранее построенных рёбер – пропустить этот вариант и те, которые он порождает, т.е. это запрет циклов.

Оптимальное решение ЗКВ полным перебором возможно. Достоинство метода – простота реализации и гарантия нахождения лучшего пути. Однако с ростом количества узлов время растёт факториально, на расчёт пути для 60 узлов уйдёт 6 триллионов лет. Поэтому идея является не лучшей для реализации.

Метод ближайшего соседа (жадный алгоритм). *Жадный алгоритм* – это алгоритм нахождения кратчайшего расстояния путём выбора ребра с наименьшим весом из тех ребер, по которым можно идти, если оно не образует

цикла с уже выбранными. Можно запуститься из любой вершины и быстро закончить алгоритм. Достоинства – простота реализации и высокая скорость работы, прирост времени выполнения медленный. Но решения, предлагаемые алгоритмом, не оптимальны. И есть ситуации, где на последних шагах приходится расплачиваться за жадность, в результате получится длиннейший путь. Алгоритм перебирает ребра в порядке возрастания весов. Он будет проверять, что добавляемые ребра удовлетворяют условиям: а) при добавлении ребра (не завершающего) не образуется цикл; б) добавленное ребро не будет третьим, выходящим из вершины.

Метод ветвей и границ, разработанный Литтлом, Мерти, Суини, Кэроллом в 1963 г. – это улучшенный перебор, отбрасывающий на каждом шаге явно неоптимальные решения. Если вершины i, j не связана между собой, то элементу матрицы приписывается длина минимального пути между i и j через другие вершины. Все перебираемые варианты делят на классы и получают оценки для них, чтобы отбрасывать варианты не по одному, а целыми классами. Достоинство – легко программируется и реализуется на любой итерации. Недостаток – трудно найти такое разделение на классы (ветви) и такие оценки (границы), чтобы процедура была эффективной по времени.

Динамическое программирование – это способ решения сложных задач, выглядящих как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. Чтобы решить задачу, требуется решить отдельные ее части только по одному разу, после чего объединить решения подзадач в одно общее решение. Это сократит количество вычислений и время выполнения. Подзадачи решаются делением, пока не приходят к тривиальному случаю задачи, решаемой за константное время.

Динамическое программирование по маскам – один из вариантов решения ЗКВ. Пусть зафиксирована начальная вершина s. Идет поиск гамильтонова цикла наименьшей стоимости от s до s, проходящего по всем вершинам один раз. Состояние динамики – это пара <вершина, маска>. Вводятся обозначения: $mask_i$ – значение i-ого бита в векторе mask; d[i][mask] –

уже найденная, наименьшая стоимость пути из i в 0, проходящего единожды по всем тем j, для которых $mask_j = 1$ (если $mask_j = 0$,то эти вершины еще не посещены). Решение требует $O(2^n*n)$ памяти и $O(2^n*n^2)$ времени. Чтобы восстановить путь, нужно воспользоваться соотношением $d[i][mask] = w(i,j) + d[i][mask – 2^j]$, для всех ребер, входящих в минимальный цикл.

Генетический алгоритм – это алгоритм поиска, используемый для задач оптимизации и моделирования, основанный на механизмах естественного отбора и наследования биологической эволюции. ГА возникли в результате попыток копирования естественных процессов живой природы. Идеи эволюционных вычислений стали появляться в 50-ых годах XX века с работ Нильса Баричелли, а в 1975 г. вышла основополагающая книга Дж. Холланда **—** «Адаптация в естественных и искусственных системах», в которой был предложен генетический алгоритм для задач оптимизации.

Достоинства ГА:

1. Унифицированный подход к решению различных проблем.
2. Отличные результаты при решении сложных NP-полных задач.
3. Могут оптимизировать решение другого метода.

Основные недостатки ГА:

1. Преждевременная сходимость.
2. Нахождение субоптимальных решений вместо оптимальных.
3. Чувствителен к настраиваемым параметрам и сложен в реализации.

Наиболее приспособленные особи получают возможность "воспроизводить" потомство с другими особями. Новое поколение будет содержать более высокое соотношение хороших характеристик. Эти характеристики распространятся по всей популяции, и популяция будет сходиться к оптимальному решению задачи. Менее приспособленные особи с меньшей вероятностью смогут воспроизвести потомков, их свойства будут постепенно исчезать в процессе эволюции. Упрощенная схема работы ГА:

1. Генерация начальной популяции.
2. Оценка популяции и её селекция (отбор в популяцию родителей).

3. Применение генетических операторов (кроссинговер, мутация, инверсия) к родительской популяции и получение потомков.

4. Генерация новой популяции (для следующей итерации).

5. Проверка условия остановки.

6. Вывод наилучшей особи.

Муравьиный алгоритм – один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра. Первая версия алгоритма, предложенная доктором наук Марко Дориго в 1992 году, была направлена на поиск оптимального пути в графе. Временная сложность муравьиного алгоритма – $O(t*m*n^2)$, где t – количество итераций, m – количество муравьев, n – число вершин.

Муравьи используют окружающую среду как средство общения, взаимодействуя через через химическое вещество – феромон, откладываемое на пройденном пути. Первоначально муравьи ходят в случайном порядке. По нахождению источника пищи они возвращаются в свою колонию, прокладывая за собой феромонами тропы.

А далее, при выборе направления муравей исходит не только из желания пройти кратчайший путь, но и из опыта других муравьев – концентрации феромонов на ребрах. Как только источник пищи обнаружен, поведение становится организованным, и все больше муравьев следует к пище тем же кратчайшим путем. И они укрепляют тропу на обратном пути, если нашли источник. Со временем феромонная тропа начинает испаряться, уменьшая свою привлекательную силу. Если бы феромоны не испарялись, то первый путь был бы самым привлекательным. Чем больше времени требуется для прохождения пути, тем сильнее испарится феромонная тропа. На коротком пути плотность феромонов останется высокой, и другие муравьи скорее всего пойдут по этому пути. Эта система называется стигмергия – механизм взаимодействия «животное-животное».

Достоинства муравьиного алгоритма:

1. Эффективный поиск рациональных решений для графовых задач.

2. Опирается на память обо всей колонии, не только о прошлом поколении.

3. Меньше подвержен неоптимальным начальным решениям.

4. Может использоваться в динамических приложениях.

5. Применялся к множеству различных задач.

Недостатки:

1. Теоретический анализ затруднён.

2. Сходимость гарантируется, но время сходимости не определено.

3. Обычно необходимо применение дополнительных методов.

4. МА сильно зависят от настроечных параметров.

Практическая часть. Все 6 алгоритмов сравниваются по времени выполнения, количеству и сложности кода, правильности результата, и зависимости результата от размерности графа. Результаты выводятся на экран таким образом:

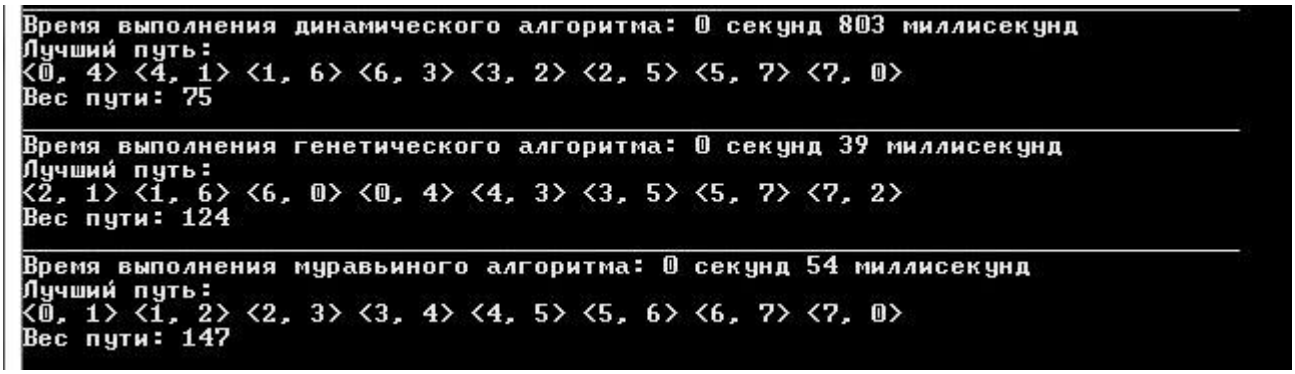

Рисунок 1 – Результаты выполнения алгоритмов на графе с 8 вершинами.

При тестировании в файле создается граф заданной размерности, со случайными весами ребер, с помощью матрицы смежности. Обязательное условие – граф связный, это проверяется при создании. Тот факт, существует ли в графе хоть один гамильтонов путь, проверяется самими методами.

Реализация полного перебора. Максимально возможное число гамильтоновых циклов для полного графа – (n-1)!/2. Метод NextPermutation создает (n-1)! возможных перестановок без повторений. Метод CyclesSearch вызывает NextPermutation с параметром v – вершиной, от которой ищутся циклы, и создает (n-1)!/2 возможных циклов от v.

После CyclesSearch, метод StartPathesGeneration удаляет из сгенерированных возможных циклов те, которых в данном графе нет, и формирует словарь, в котором каждый путь имеет порядковый номер. Параметр parallel = true означает запуск генерации циклов с помощью Parallel.Invoke(), а false – обычный запуск. Воспользовавшись словарем со сгенерированными путями, метод Search ищет в словаре путь минимального веса, и возвращает индекс этого пути. Распараллеливание этого алгоритма дало ускорение примерно на 6-8%.

Реализация жадного алгоритма. Для рассмотрения был взят метод ближайшего соседа, он очень напоминает поиск минимального остова. Каждый раз из смежных с вершиной ребер берется ребро с минимальным весом, записывается в список. Метод отмечает, что вершина пройдена, и повторяет от неё те же самые действия. Когда метод доходит до исходного города, который уже был помечен как пройденный, и все вершины графа в этот момент уже пройдены – путь готов. Метод FindPath занимается собственно поиском пути (в FindPathParallel, используется запуск с помощью Parallel.Invoke). NearestNeighbourAlgo инициализирует матрицу посещений вершин и вызывает FindPath от заданной v. Эффект замедления от распараллеливания этого алгоритма составил в среднем 50%.

Реализация метода ветвей и границ. Функция RBoundCalculation выполняет первый этап алгоритма – вычисляет константы приведения для столбцов и строк и переходит к новой матрице затрат. R – это так называемая «нижняя граница», вычисляется заново при каждом вызове, и ее значение приближается к длине минимального цикла.

Далее вызывается MaxFineCalculation – она вычисляет «штраф» за неиспользование каждого ребра, в ячейке которого в матрице после преобразований вышел 0. Для всех нулевых элементов вычисляются штрафы, среди них выбирается максимум. Если максимумов несколько, для каждого по очереди производятся дальнейшие действия. Метод Cancellation вычеркивает из матрицы строку и столбец с индексом максимума. SetInfinity устанавливает

int.MaxValue в ячейку на пересечении столбца и строки, в которых знака бесконечности нет, чтобы избежать замыкания контура раньше времени.

Все вызовы данных функций происходят в основном методе Start, в котором и осуществляется рекурсия. Все найденные пути хранятся в словаре, и как только соответствующее число путей найдено, их можно вывести на экран. Параллелизм был достигнут путем использования Parallel.Invoke и ускорил оригинальный метод приблизительно на 5%.

Реализация генетического алгоритма

Генерация начальной популяции производится случайным образом. Но пути обязательно должны быть корректными, то есть связными. Случайность заключается в том, что нужно взять абсолютно произвольную часть области допустимых решений. За инициализацию начальной популяции отвечает CreateBeginPopulation, использующий GrayCodeCreation для создания кода Грея, которым будет зашифрован каждый город. Здесь используется метод NextPermutation для создания перестановок. Соответственно, есть Decoding для расшифровки кода Грея.

Пусть для графа до 10 вершин размер популяции будет равен 50 маршрутам, от 10 до 20 вершин – 100 маршрутов, а для более 20 вершин в начальной популяции будет 500 маршрутов. Чтобы взять эту произвольную часть области допустимых решений, нужно использовать механизм шаблонов, который позволит создать корректные циклы без полного перебора вариантов. Шаблон – это последовательность, в которой некоторые города уже известны (и связны), а на месте остальных стоят *. Вместо * можно рандомом подставлять номера оставшихся городов и так генерировать пути. Механизм не имеет сложных вычислений и работает на основе перестановок.

За условие остановки отвечает метод StopConditionPerformed. Было решено скомбинировать 2 условия – нахождение оптимума и достижение определенного числа итераций. Crossbreeding, Mutation, Inversion отвечают за выполнение генетических операторов. UseGeneticOperators высчитывает, сколько особей должны быть подвержены скрещиванию, мутации и инверсии;

вызывает эти операторы; «чистит» список родителей от тех, кто уже прошел воздействие операторов. FitnessAssessmentForOne и FitnessAssessmentMiddle высчитывают приспособленность особи и среднюю приспособленность по популяции. Selection – отвечает за «естественный отбор», турнирную селекцию.

Реализация муравьиного алгоритма

Если в пути муравья какая-то вершина не соединяется с последующей вершиной, такая ситуация называется тупиком. Первый способ решения – это оставить данного муравья в том месте, откуда он пытался выйти, феромон не обновлять, запустить следующего. Теряется возможный вариант пути, но это увеличивает быстродействие. Второй способ – сделать комбинаторный возврат и пойти в другую вершину из предыдущей; это хорошо работает на задаче построения простого пути, там отсечь тупик можно за полиномиальное время. В гамильтоновом цикле так сделать не получится, потому что алгоритм на каком-то этапе может пойти в обход дерева комбинаторного перебора с факториальным числом ветвей. Поэтому был выбран первый вариант.

В классе AntOptimization содержится весь функционал, необходимый для этого метода. Класс EdgeForAnt хранит кроме сведений о начале, конце и весе ребра, его видимость и след феромона. Ant реализует муравья, у которого есть вершина его местонахождения, порядковый номер, маршрут и т.д.. EdgesListFilling заполняет список ребер, соответствуя EdgeForAnt. StopCondition делает проверку на условие остановки, ограничиваясь итерациями либо ситуацией, когда длина пути достигла оптимума.

Метод Start содержит в себе полный муравьиный алгоритм. Start расставляет каждого муравья в каждый город случайным образом, выбирает, какие муравьи будут «элитными», присваивает начальные значения видимости и следа феромона каждому ребру, выбирает произвольный «кратчайший» маршрут. Каждый муравей идет в тот город, у которого для этого муравья наибольшая вероятность захода, и таким образом строит путь. След феромона обновляется в соответствии с формулой испарения, и с проходом муравья – обычного или элитного. На каждой итерации алгоритм ищет, у какого муравья

получилось построить наиболее короткую дорогу. Этот процесс происходит до выполнения условия остановки.

Реализация динамического программирования. В классе Dynamic – 3 основных функции: Restore, Calc и Start. Calc выполняет поиск самого «дешевого» перехода из заданной вершины в следующую не пройденную, используя побитовые сдвиги и маски. Если маска полностью заполнена единицами (это значит, что вершина присутствует в пути), путь найден. Restore – рекурсивный метод, который ищет сам путь, вызывая Calc от каждой вершины. Если Calc выяснил, что от текущей вершины в k-тую вершину путь самый короткий, Restore кладет k-тую вершину в путь. Но Calc может найти неверный путь, проходящий по всем вершинам, но в неправильном порядке. В таком случае этот путь исключается. Start преобразует матрицу смежности графа к нужной форме (в массиве Z), вызывает методы; если путь успешно найден, Start переведет его из маски в List<edge> и возвратит.

Общий сравнительный анализ методов. Все методы прошли тестовый запуск на графах от 8 вершин до максимального числа, которое охватывал алгоритм. Для каждого числа вершин по каждому алгоритму было проведено 10 тестов, результат по времени усреднен, по весу записан последний результат. Характеристики процессора, на котором проходили тесты:

Имя ОС	Microsoft Windows 7 Максимальная
Версия	6.1.7601 Service Pack 1 Сборка 7601
Изготовитель	Hewlett-Packard
Модель	HP Pavilion dv6 Notebook PC
Тип	x64-based PC
Процессор	Intel(R) Core(TM) i3-2330M CPU @ 2.20GHz, 2200 МГц, ядер: 2, .
Версия BIOS	Insyde F.2B, 29.05.2014 логических процессоров: 4
Полный объем физической па...	3,90 ГБ
Доступно физической памяти	1,11 ГБ
Всего виртуальной памяти	7,79 ГБ
Доступно виртуальной памяти	4,61 ГБ

Рисунок 2 – Характеристики процессора.

Для удобства просмотра результатов результаты были сведены в таблицы: одна иллюстрирует стоимость пути, другая – время выполнения.

После сбора результатов алгоритмов можно представить практический анализ. Исследуются 4 аспекта: время выполнения, количество и сложность кода, точность результата, и зависимость результата от размерности графа.

1. *Время выполнения*

Безоговорочным лидером по времени выполнения стал жадный алгоритм – он искал первый попавшийся путь, не рассматривал множество других вариантов прохождения по графу. Но распараллеливание увеличило время выполнения простого кода в 2-2,5 раза. Среда разработки затратила больше сил и времени на распараллеливание, чем на поиск решения, это не оптимально.

На втором месте стоит метод ветвей и границ – сложные математические вычисления оправдали себя. Распараллеливание алгоритма повлияло на время работы неоднозначно – в некоторых случаях было ускорение на 5-10%, но в других случаях либо разницы между параллельной и последовательной версией не было, либо параллелизм замедлил алгоритм – это происходило, если в графе было немного ребер.

На третье место можно поставить генетический алгоритм – несмотря на количество итераций и приближенность результата, его время работы в целом не превысило 1-3 секунд. Далее следует муравьиный алгоритм – для 30 вершин он затрачивал 4 секунды и более, в то время как ГА не превысил 2 секунд.

Динамическое программирование заняло предпоследнее место, алгоритм имеет временную сложность, близкую к факториальной. Метод является улучшением по сравнению с полным перебором, но очень небольшим – для 20 вершин время выполнения достигло почти 15 секунд, а далее замерить не получилось, т.к. стало не хватать памяти для динамики с побитовыми сдвигами.

Худший результат по времени выполнения дал полный перебор – его время возрастало факториально (с 1 секунды сразу до 40), и уже для 11 вершин в текущей реализации дождаться результата не удалось.

2. *Количество и сложность кода*

Меньше всего строк кода потребовалось на полный перебор, жадный алгоритм и динамическое программирование. Каждый из методов был реализован в 2-3 функции, и их код несложен и интуитивно понятен.

Муравьиный алгоритм занял больше не только по размеру кода, но и по сложности – в нем уже используются в достаточно большом количестве пересчеты коэффициентов, таких, как видимость ребра, испарение феромонов, количество наносимых на ребро феромонов, и т.д. Также требуется запоминание параметров муравья: номер, список пройденных городов и т.д.

Самыми сложными в результате анализа были признаны метод ветвей и границ и генетический алгоритм. ГА реализует сложную схему естественного отбора по подобию происходящего в живой природе, он требует специального кодирования городов. Также ГА имеет достаточно объемный код, в котором много проверок на крайние случаи, и более десятка функций; алгоритму необходимо выполнить несколько десятков или сотен итераций, как и МА, чтобы добиться хотя бы какого-нибудь.

Метод ветвей и границ затрагивает много сложных математических вычислений с использованием матриц, проверки, пересчеты коэффициентов.

3. Правильность (корректность) результата

По корректности результата, полный перебор и динамическое программирование – «эталонные» точные методы, они дают наименьший вес пути, и с ними можно сравнивать результаты других методов. В жадном алгоритме вес пути во всех случаях отличался от наименьшего веса лишь на 10-20 единиц. И метод ветвей и границ показал почти такие же результаты – разница с минимумом была в 30-40 единиц.

ГА выдавал вес пути на 100-200 единиц отличающийся от минимума. Это можно обосновать тем, что начальная популяция состояла всего из 50 или 100 случайных особей, в то время как число возможных путей измерялось десятками или сотнями тысяч. Ее размер можно было бы выбрать и лучшим образом, тогда возросла бы вероятность найти хорошее решение.

Результат МА отличался от минимума и на 200, и на 300 единиц. Так как муравьи не пытались найти другое решение при попадании в тупик, а возвращались в исходную вершину в случае «неудачи», многим хорошим ребрам было случайно присвоено низкое значение феромона. Для исправления этого недоразумения требуется реализовать возврат по рекурсивному дереву на шаг назад в каждом случае, когда муравей попадает в тупик. Этот подход не был реализован в силу сложности реализации, и объемности самого алгоритма.

4. Зависимость результата от размерности графа.

Лучше всего проявил себя в разрезе независимости от размерности графа жадный алгоритм. При 40 вершинах его время выполнения возросло менее, чем в 2 раза (по сравнению с 20 вершинами). Также хорошие характеристики показал метод ветвей и границ, он работал достаточно быстро на графах даже свыше 30 вершин (около 1,5 секунд).

В генетическом алгоритме даже для графа с 31 вершиной ответ был выдан за время меньше 2 секунд, что доказывает хорошие характеристики ГА для графов большой размерности. При графе с 32 вершинами размер популяции возрос до 500 особей, и такую большую популяцию данная реализация уже не смогла обработать. Но если увеличивать размер популяции не так стремительно, можно получить результаты и для 40 вершин.

МА показал средние результаты – для 40 вершин его время было уже более 23 секунд. Однако время возрастало пропорционально, и МА отработал даже на 40 узлах графа, что характеризует его как эффективный алгоритм.

Динамическое программирование уже сильно зависело от количества вершин в графе, т.к. оно потребляло много памяти для массива динамических состояний. n было установлено изначально равным 8, потом оно поднялось до 12, 16, и наконец до 20. Увеличить массив далее было невозможно, так как алгоритм вылетал по исключению «переполнение памяти».

Полный перебор – это наглядная иллюстрация того, что точный ответ дается большими временными затратами. Уже для 10 вершин дождаться результата не удалось из-за крайне большого числа переборных вариантов.

Заключение.

Задача коммивояжера имеет множество способов применения в реальной жизни, транспортных задачах, при доставке грузов и в прочих подобных ситуациях. Отсутствие единственно правильного, оптимального метода решения этой задачи и попытки его создать привели к появлению множества методов, которые заметно отличаются и по времени выполнения, и по оптимальности выбранного пути. Выбор конкретного метода осуществляется в зависимости от критерия, который важен в конкретном случае, и количества городов или других объектов, которые связаны между собой путями.

Из обширного списка методов решения, в теоретической части были рассмотрены и в практической части реализованы на языке C# 6 методов. Генетический алгоритм, муравьиный алгоритм, метод ветвей и границ и жадный алгоритм являются приближенными методами, а полный перебор и динамическое программирование дают точный результат.

Было проведено тестирование методов на случайных графах от 8 до 40 вершин, замерено время их выполнения. Проведенный сравнительный анализ по показателям эффективности дал понять, какую оптимальность имеет каждый из методов, какие у них преимущества и недостатки. Лучшими по времени выполнения были признаны жадный алгоритм и метод ветвей и границ. Генетический алгоритм также дал хороший результат по времени, муравьиный отработал на достаточно большом числе вершин. Лучшие по точности – полный перебор и динамическое программирование.