Санкт-Петербургский политехнический университет
Высшая школа прикладной математики и вычислительной физики, ФизМех

Направление подготовки
«01.03.02 Прикладная математика и информатика»
Специальность «Системное программирование»

Курсовая работа
тема "Сравнительный анализ реализации задачи коммивояжёра с усложнением"
дисциплина "Методы оптимизации"

Выполнили студенты гр. 5030102/00201 Гвоздев С.Ю.,
(Алгоритм иммитации отжига)
Золин И.М.
(Формулировка задачи и её формализация, Муравьиный алгоритм)
Хламкин Е.М.
(Генетический алгоритм)

Преподаватель: Родионова Е.А.

Санкт-Петербург

2022

Содержание

1 Введение

Задача коммивояжера (TSP -Travelling Salesman Problem) — одна из наиболее активно изучаемых задач вычислительной математики.

Эта задача состоит в том, чтобы найти кратчайший путь , по которому коммивояжер должен пройти через список городов и вернуться в исходный город. Приведен список городов и расстояние между каждой парой.

TSP полезен в различных приложениях в реальной жизни, таких как планирование или логистика. Например, менеджер концертного тура, который хочет запланировать серию выступлений для группы, должен определить кратчайший путь для тура, чтобы обеспечить сокращение транспортных расходов и не утомлять группу без необходимости.

Это NP-сложная задача. Проще говоря, это означает, что вы не можете гарантировать нахождение кратчайшего пути в разумные сроки. Однако это не уникально для TSP.

2 Формулировка задачи и её формализация

2.1 Формулировка задания

Рассматривается следующая формулировка задачи коммивояжера. Есть N городов, соединённых между собой односторонними дорогами. Коммивояжер выезжает из начального города, он должен посетить остальные N-1 городов (существует также формулировка, при которой необходимо вернуться обратно в стартовый город, однако в рамках данной работы ограничимся "незамкнутым"вариантом). Повторное посещение городов запрещено.

Дополнительно: каждый город характеризуется рейтингом, за заданное время нужно обойти и набрать максимально возможный суммарный рейтинг.

2.2 Формализация задания

$n \geq 0$ - количество городов
$M_{n,n}$ - матрица времени, $m_{ij} \geq 0; i, j = \overline{1, n}$
$S \geq 0$ - ограничение по времени
C_n - массив рейтингов городов, $c_i \geq 0; i = \overline{1, n}$

$$\begin{cases} \sum_{i=1}^{k} c_i \to max; k = \overline{1, n} \\ \sum_{i=1,j=1}^{k} m_{ij} \leq S \end{cases} \tag{1}$$

(1) : Множество ограничений:

1. Функция цели (задача поиска максимума, суммарный рейтинг)
2. Ограничения (по времени)

3 Решение задачи

3.1 Алгоритм полного перебора

Метод полного перебора - самый наивный и времязатратный алгоритм поиска оптимального пути задачи коммивояжера с временными ограничениями, однако единственный, который всегда находит наиболее оптимальный путь.

Временная сложность алгоритма: O(n!)

Алгоритм метода: перебор осуществляется на заданных начальных данных (начальный город, ограничение по времени и псевдослучайная матрица времён) путём последовательной выборки наименьшего по индексу города из ещё непройденных. При включении города в текущий путь к рейтингу пути добавляется значение рейтинга, соответствующее новому городу. Далее функция поиска подходящего города вызывается снова, уже считая только что найденный город изначальным. Рекурсия происходит до тех пор, пока не кончится отведённое на переходы между городами время. Изначально максимальный рейтинг пути задан как 0 (поэтому даже в худшем случае, когда ограничение времени меньше всех путей из города, рейтинг пути будет больше 0 и равен рейтингу 1-го города). Если за время, меньшее ограничения, алгоритму удалось найти более благоприятный путь, переменная, отвечающая за максимальный рейтинг пути, изменяет своё значение на свеженайденное, что в конце концов и приводит к решению поставленной задачи.

3.2 Муравьиный алгоритм

3.2.1 Введение

Первоначально идею муравьиного алгоритма предложил Марко Дориго в 1992 году в его докторской диссертации, первый алгоритм был направлен на поиск оптимального пути в графе, основанном на поведении муравьев, ищущих путь между своей колонией и источником пищи для решения задач оптимизации.

В естественном мире муравьи бродят хаотично и, найдя пищу, возвращаются в свою колонию, прокладывая феромонные тропы. Другие же муравьи, находя тропы с феромонами, используют эту информацию, тем самым укрепляя феромонную тропу(положительная обратная связь).

(*):Чем короче путь до источника пищи, тем меньше времени понадобится на перемещение муравьям - следовательно, тем быстрее оставленные на нем следы будут заметны.

(**): В итоге, чем больше муравьев проходит по определенному пути, и чем этот путь короче, тем он становится более привлекательным для остальных муравьев.

3.2.2 Описание

Каждый муравей рассматривается как отдельный и независимый коммивояжер, решающий свою собственную проблему. В течение одной итерации муравей проходит весь маршрут коммивояжера.

"Случайность"реализует вероятностное правило, с пмощью которого обеспечивается положительная обратная связь: вероятность того, что ребро графа включено в маршрут муравья, пропорционально его значению феромона.

Чтобы имитировать поведение муравьев (**), объем виртуальных феромонов, размещенных на ребре графика, принимается обратно пропорциональным длине пути.

(*): Однако положительная обратная связь приводит к застою: в этом случае все муравьи выбирают один неоптимальный путь. Чтобы избежать этого, существует отрицательная обратная связь: через ферромон вводится испарение. Интенсивность ис-

парения не должна быть слишком высокой; в противном случае область поиска сузится(ситуации, когда колония преждевременно “забывает” свой опыт, полученный в прошлом (потеря памяти)).

Для каждого муравья проход из города i в город j зависит от следующих трех компонентов: список запретов (tabu list) или память муравья, видимость и следы виртуальных феромонов.

Список запретов - это структура данных, которая сохраняет список уже посещенных городов, которые не следует посещать снова. Этот список увеличивается в размерах на каждом шаге и устанавливается равным нулю в начале каждой итерации алгоритма. Обозначение: $J_{i,k}$ - список городов, которые еще предстоит посетить k-ому муравью, расположенному в i-ом городе .

Видимость - является обратной величиной к расстоянию. $\eta_{i,j} = \frac{1}{D_{i,j}}$, $D_{i,j}$ - это расстояние между городами i и j. Видимость - это локальное статическое значение, отражающее эвристическое желание переехать из города i в город j: чем ближе город, тем сильнее желание его посетить.

Муравьиный алгоритм можно классифицировать как вероятностный, то есть он дает только приближенное решение, не гарантируя его оптимальности.

3.2.3 Формализация

3.2.3.1 Общий случай

Ниже приведён алгоритм:

1. Инициализация(создаем муравьёв и задаем начальные значения феромона)
   (a) m - количество муравьев, участвующих в построении пути
   (b) n - количество городов;
   (c) Ввод $D_{n,n}$ - матрицы расстояний
   (d) Инициализация параметров алгоритма:
      i. α, β– константные параметры(описывают вес феромонного следа и посещаемость при выборе маршрута).
      ii. Q>0 – параметр, имеющий значение порядка цены оптимального решения.
      iii. $p \in [0, 1]$ – коэффициент испарения,
      iv. τ_0 - начальный уровень феромона в каждом городе
   (e) Инициализация видимости $\eta_{i,j}$ и уровня феромона $\tau_{i,j}(t)$
   $\eta_{i,j} = \frac{1}{D_{i,j}}$, $D_{i,j}; i \neq j$
   $$\tau_{i,j}(0) = \begin{cases} \tau_0, i \neq j \\ 0, i = j \end{cases}$$
2. Вероятность перемещения k-ого муравья на итерации t из города i в город j вычисляется по следующему вероятностно–пропорциональному правилу:
$$P_{i,j,k}(t) = \begin{cases} \frac{(\tau_{i,j}(t))^{\alpha}(\eta_{i,j})^{\beta}}{\sum_{l \in J_{i,k}}(\tau_{i,l}(t))^{\alpha}(\eta_{i,l})^{\beta}}, j \in J_{i,k} \\ 0, j \notin J_{i,k} \end{cases} \quad (2)$$

Когда $\alpha = 0$, выбирается ближайший город, что соответствует жадному алгоритму в классической теории оптимизации. Когда $\beta = 0$, учитывается только след феромона, что означает, что все муравьи выбирают один неоптимальный маршрут. Чтобы обеспечить хорошую динамику оптимизации, рекомендуется установить $\alpha \geq \beta$.

Отметим, что (2) определяет вероятности выбора конкретного города. Сам выбор осуществляется по принципу "колеса рулетки": каждый город на нем имеет свой собственный сектор с площадью, соответствующей вероятности (2).

3. Феромон, откладываемый k-ым муравьём, использующим ребро (i,j)

$$\Delta\tau_{i,j,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, (i,j) \in T_k(t) \\ 0, (i,j) \notin T_k(t) \end{cases} \tag{3}$$

где $T_k(t)$ - это маршрут муравья k на итерации, $L_k(t)$ – длинна маршрута(цена текущего решения для k-ого муравья)

4. Обновляем уровень феромона в соответствии с приведённой формулой

$$\tau_{i,j}(t+1) = (1-p)\tau_{i,j}(t) + \sum_{k=1}^{m} \Delta\tau_{i,j,k}(t) \tag{4}$$

3.2.3.2 Ant Colony System(ACS)

Для задачи коммивояжера правило обновления феромонов (4) принимает вид

$$\tau_{i,j}(t+1) = (1-p)\tau_{i,j}(t) + p\Delta\tau_{i,j,e}(t) \tag{5}$$

где (i, j) - край наилучшего маршрута (либо на текущей итерации, либо с начала алгоритма

$$\Delta\tau_{i,j,e}(t) = \begin{cases} \frac{Q}{L^+}, (i,j) \in T^+ \\ 0, (i,j) \notin T^+ \end{cases}$$

"Элитные"муравьи(муравьи в модификации алгоритма) выделяют феромоны только по рёбрам лучшего найденного маршрута T^+. Для задачи коммивояжера берется значение феромона "элитного"муравья на каждом ребре маршрута T^+ равным $\frac{Q}{L^+}$, где L^+ - длина маршрута T^+.

Шаг (2) модифицируется следующим образом: k-й муравей перемещается с вероятностью q_0 из города i в наиболее привлекательный город $z \in J_{i,k}$ и с вероятностью $(1-q_0)$ выбирает город j по шагу (2). Чем больше q_0, тем выше эффективность использования опыта, полученного колонией муравьев при синтезе новых маршрутов. Наиболее привлекательный город определяется как

$$z = \arg\max_{j \in J_{i,k}} (\tau_{i,j}(t))^{\alpha} (\eta_{i,j})^{\beta} \tag{6}$$

На каждой итерации, при переходе от из города i в город j муравей “съедает” некоторое количество феромонов с ребра (i – j). Это ребро теряет свою привлекательность для других муравьев, что вынуждает их рассматривать альтернативные маршруты из городов i и j. Решения становятся более разнообразными благодаря динамичному обновлению распределения феромонов.

Помимо изменений алгоритма, касающихся непосредственно модификации классической версии, введём изменения, относящиеся к отличиям решаемой задачи от классической задачи коммивояжёра.[7]

1. Уровень феромонов на рёбрах обновляется не только в конце очередной итерации, но и при каждом переходе муравьёв из узла в узел.

2. В конце итерации уровень феромонов повышается только на лучшем из найденных путей.

3. Алгоритм использует изменённое правило перехода: либо, с определённой долей вероятности, муравей выбирает лучшее – в соответствие с рейтингом и уровнем феромонов – ребро, либо производит выбор так же, как и в классическом алгоритме.

4. Наконец, учитывая введённые дополнительные ограничения задачи, будем обнулять вероятность перехода в вершину, если оставшееся время перехода T_k k-того муравья меньше стоимости перехода. Также, поскольку решается задача максимизации, лучший путь выбирается по суммарному рейтингу, а величины $\eta_{i,j}$ учитывают рейтинг узла j.

3.2.4 Тестовый пример

3.2.4.1 Начальные данные

$$\begin{bmatrix} 0 & 14 & 36 & 34 & 22 \\ 14 & 0 & 19 & 26 & 30 \\ 36 & 19 & 0 & 15 & 36 \\ 34 & 26 & 15 & 0 & 21 \\ 22 & 30 & 36 & 21 & 0 \end{bmatrix}$$ Матрица - времени

Массив рейтинга - [75 35 45 5 25]

3.2.4.2 Начальные значения параметров

- Количество поколений: 1
- Количество муравьёв в поколении: 2
- Коэффициент запаха $\alpha = 0.1$
- Коэффициент расстояния $\beta = 2$
- Глобальный коэффициент обновления e = 0.1
- Локальный коэффициент обновления p = 0.1
- Количество выпускаемых феромонов Q = 1
- Коэффициент баланса перехода между городами q = 0.9

3.2.4.3 Алгоритм

1. Для начала проинициализируем каждый элемент матрицы феромонов начальным значением $\tau_{i,j}(0) = \phi = \frac{Q}{n \cdot 2} = \frac{1}{10}$, а затем обнулим диагональные элементы.

2. Затем для каждого из муравьёв подсчитаем вероятности перехода в k-тый город из первого города по формуле из пункта 2 алгоритма выше. Для тех городов, где муравей уже был, вероятность считаем равной 0, также равной 0 будет вероятность перехода в k-тый город, если расстояние до него больше, чем оставшееся время муравья.

   Пример расчета вероятности перехода первого муравья из первого города во второй: $P_{10,1} = \frac{(\tau_{1,2}(t))^{\alpha}(\eta_{1,2})^{\beta}}{max} = \frac{(\frac{1}{10})^1 + (\frac{1}{14})^2 \cdot 35}{75} = 2.38095e - 4$

   Таблица вероятностей переходов для первого муравья

№ вершины	1	2	3	4	5
1	0	2.38095e-4	4.62963e-5	5.76701e-6	6.88705e-5
2	0	0	1.66205e-4	9.86193e-6	3.7037e-5
3	0	0	0	2.96296e-5	2.64550e-4
5	0	0	0	0	0

Таблица сгенерированных $rand_1$:

1	2	3
0.568823660872193	0.469390641058206	0.337122644398882

3. На каждой итерации сгенерируем случайное число $rand_k$ лежащее от 0 до 1. Если все вероятности перехода для данного муравья равны 0, значит муравей закончил свой путь, и мы переходим к следующему. Иначе, если случайная величина соответствующая муравью меньше или равна коэффициенту q = 0.9, то муравей переходит в вершину с наибольшей вероятностью, время этого муравья уменьшается на величину расстояния от текущей вершины до той, в которую он перешёл. Если случайная величина больше q, то в качестве вершины перехода берётся случайная вершина, для которой верно, что вероятность перехода в неё больше 0. В этом случае точно так же время этого муравья уменьшается на величину расстояния от текущей вершины, до той в которую он перешёл.

   Итоговый суммарный рейтинг пути первого муравья равен

   $S_1 = 75 + 35 + 45 + 25 = 180$

   - Path = [1,2,3,5]
   - Rating = 180

4. Локально обновляем феромон на пути первого муравья по формуле

   $\tau_{i,j}(1) = (1-p)\tau_{i,j}(0) + p\phi = (1-0.1)\frac{1}{10} + 0.1\frac{1}{10} = 0.1$

   Так как мы выбрали $\phi = \frac{Q}{n \cdot age}$,то значения остались теми же.

5. Аналогично "запустим"второго муравья. Таблица вероятностей переходов для второго муравья

№ вершины	1	2	3	4	5
1	0	2.38095e-4	4.62963e-5	5.76701e-6	6.88705e-5
2	0	0	0.5613	0.2450	0.1344
4	0	0	2.66667e-4	0	7.55858e-5
3	0	0	0	0	0

6. Таблица случайных значений для второго муравья, на 2-ем шаге значение случайного числа превысило 0.9, значения вероятностей были нормированы и случайным образом из них было выбрано 4 значение.

   Итоговый суммарный рейтинг пути второго муравья равен

   $S_2 = 75 + 35 + 5 + 45 = 160$

   - Path = [1,2,4,3]
   - Rating = 160

Таблица сгенерированных $rand_2$:

1	2	3	4
0.794284540683907	0.913337361501670	0.337122644398882	0.414233540182197

Максимальное значение рейтинга на первой итерации: 185. Глобальное обновление феромонов не нужно, так как использовалось всего одно поколение. Алгоритм закончил свою работу.

7. Итого, получаем:
   - Path = [1,2,3,5]
   - Rating = 180

3.3 Алгоритм иммитации отжига

3.3.1 Описание

Как и многие гениальные вещи, данный метод был подсмотрел у матушки природы. За основу взят процесс кристаллизации вещества, который в свою очередь «приручили» хитрые металлурги, для повышения однородности металла. У каждого металла есть кристаллическая решетка, если говорить грубо, она описывает геометрическое положение атомов вещества. Совокупность позиций всех атомов будем называть состоянием системы, каждому состоянию соответствует определенный уровень энергии. Цель отжига – привести систему в состояние с наименьшей энергией.

В ходе «отжига» металл сначала нагревают до некоторой температуры, что заставляет атомы кристаллической решетки покинуть свои позиции. Затем начинается медленное и контролируемое охлаждение. Атомы стремятся попасть в состояние с меньшей энергией, однако, с определенной вероятностью они могут перейти и в состояние с большей. Эта вероятность уменьшается вместе с температурой. Переход в худшее состояние, часто помогает в итоге отыскать состояние с энергией меньшей, чем начальная. Процесс завершается, когда температура падает до заранее заданного значения.

Такая сложная схема с вероятностями переходов из точки в точку нужна потому что, эвристические алгоритмы имеют склонность к зацикливанию на локальном минимуме (максимуме) и выдаче его за глобальный оптимум. Чтобы выйти из такой ситуации, необходимо время от времени повышать энергию системы. При этом общая тенденция к поиску наи-меньшей энергии сохраняется. В этом и состоит суть метода имитации отжига.

3.3.2 Формализация

3.3.2.1 Общий вид алгоритма

1. Случайным образом выбирается начальная точка $x = x_0; x_0 \in S$. Текущее значение энергии E устанавливается в значение $f(x_0)$.
2. k-я итерация основного цикла состоит из следующих шагов:
   (a) Сравнить энергию системы E в состоянии x с найденным на текущий момент глобальным минимумом. Если $E = f(x)$ меньше, то изменить значение глобального минимума.
   (b) Сгенерировать новую точку $x_l = G(x; T(k))$
   (c) Вычислить значение функции в ней $E_l = f(x_l)$
   (d) Сгенерировать случайное число α из интервала [0:1]

(e) Если $\alpha < h(E_l$ - $E; T(k))$, то установить $x \leftarrow x_l; E \leftarrow E_l$ и перейти к следующей итерации. Иначе повторить шаг (b), пока не будет найдена подходящая точка x_l.

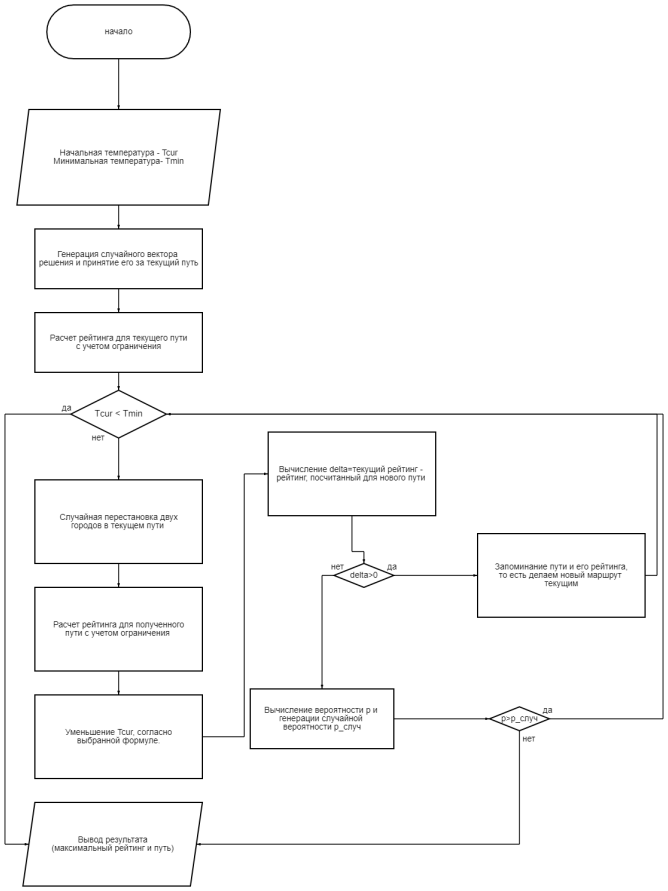
$h((\Delta E); T)$- функция вероятности принятия.

3.3.2.2 Алгоритм, адаптированный под поставленную задачу

T_{cur} - Текущая температура на каждой итерации. T_{min} – Минимальная температура.

1. $i = 0$. Генерируется случайный путь S_0, и для него считается рейтинг R_0 с учетом ограничения по времени.
2. Если $T_{cur} > T_{min}$, переходим к шагу 3, иначе переходим к шагу 12
3. Переставляются два случайных города в последовательности S_i
4. i += 1
5. Считается рейтинг для S_i
6. $\Delta = R_i$ - R_{i-1}
7. Если $\Delta > 0$, то переходим к шагу 11, иначе переходим к шагу 8
8. Генерируется случайная вероятность p_{rand}
9. Вычисляется вероятность p = exp(-Δ/T_{cur})
10. Если $p \geq p_{rand}$, то S_i заменяется на S_{i1} и R_i заменяется на $Ri1$
11. Переходим к шагу 2
12. Заканчиваем алгоритм и результатом будет S_i и R_i

3.3.2.3 Блок схема, адаптированного алгоритма.



3.3.3 Тестовый пример

$$\begin{bmatrix} 0 & 14 & 36 & 34 & 22 \\ 14 & 0 & 19 & 26 & 30 \\ 36 & 19 & 0 & 15 & 36 \\ 34 & 26 & 15 & 0 & 21 \\ 22 & 30 & 36 & 21 & 0 \end{bmatrix}$$ Матрица - времени

Массив рейтинга - [75 35 45 5 25]
$S = 70$ - ограничение по времени
$T_0 = 80$
$T_{min} = 3$
T_n=0.5*T_{n-1}

1. $path_1$ = [1 5 3 2 4]
   CurrentTime = 22 + 36 = 58
   CurrentRating = 75 + 25 + 45 = 155
   $R_{max} = 155$
   $T_1 = 80 > T_{min}$

2. $path_2$ = [1 4 3 2 5]
   CurrentTime = 34 + 15 + 19 = 68
   CurrentRating = 75 + 5 + 45 + 35 = 160
   $R_{max} = 160$
   $T_2 = 40 > T_{min}$

3. $path_3$ = [1 4 2 3 5]
   CurrentTime = 34 + 26 = 60
   CurrentRating = 75 + 5 + 35 = 115 → $CurrentRating$ <R_{max}
   $p^* = e^{\frac{-(160-115)}{20}} = e^{-2} = 0.105$, пусть $p_{rand} = 0.1$ →$p^* > p_{rand}$ $T_3 = 20 > T_{min}$

4. $path_4$ = [1 4 3 5 2]
   CurrentTime = 34 + 15 = 49
   CurrentRating = 75 + 5 + 45 = 125 → $CurrentRating$ <R_{max}
   $p^* = e^{\frac{-(160-125)}{10}} = e^{-3.5} = 0.03$, пусть $p_{rand} = 0.002$ →$p^* > p_{rand}$ $T_4 = 10 > T_{min}$

5. $path_5$ = [1 2 3 5 4]
   CurrentTime = 14 + 19 + 36 = 69
   CurrentRating = 75 + 35 + 45 + 25 = 180
   $R_{max} = 180$
   $T_5 = 5 > T_{min}$

6. $path_6$ = [1 2 3 4 5]
   CurrentTime = 14 + 19 + 15 + 21 = 69
   CurrentRating = 75 + 35 + 45 + 25 + 5 = 185
   $R_{max} = 185$
   $T_6 = 2.5 < T_{min}$ ⇒ Заканчиваем итерационный процесс .
   На выходе получаем:
   $Path$ = [1, 2, 3, 4, 5]
   $Rating$ = 185

3.4 Генетический алгоритм

3.4.1 Введение

Генетический алгоритм – это эвристический алгоритм поиска, используемый для решения задач оптимизации путем случайного подбора, комбинирования и вариации искомых параметров с применением механизмов, напоминающих биологическую эволюцию. Применение алгоритма выходит далеко за рамки решения одной лишь задачи Комивояжера. К примеру, его можно встретить в программах обучения нейронных сетей [8]. Остановимся же на приложении генетического алгоритма именно к задаче Комивояжера.

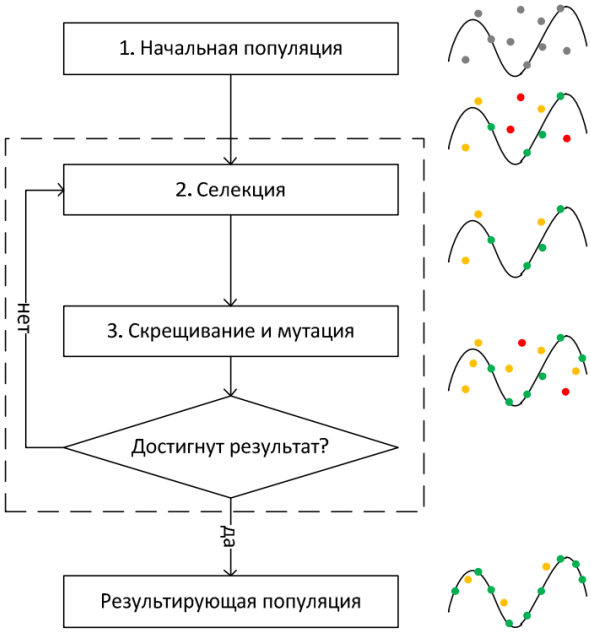
3.4.2 Описание

При описании алгоритма удобно использовать естественную терминологию. Поэтому некоторые понятия нашей задачи переформулируем следующим образом:

1. Хромосома – последовательность вершин, которые образуют маршрут
2. Ген - вершина
3. Популяция – множество хромосом (маршрутов)
4. Особь – набор хромосом, удовлетворяющих решению
5. Скрещивание – операция, при которой две хромосомы обмениваются своими частями
6. Мутация – случайное изменение одной или нескольких позиций в хромосоме

3.4.3 Классический алгоритм

Рассмотрим классический генетический алгоритм. Для лучшего понимания проиллюстрируем описание схемой:



Шаги алгоритма:

1. Формируется “нулевая” популяция – выбирается и строится фиксированное количество потенциальных решений задачи. Чаще всего построение происходит произвольным образом, однако существуют модификации алгоритма, при которых стартовый набор хромосом выбирается исходя из определенных свойств задачи. Мы будем использовать произвольный вариант. Далее каждому из маршрутов ставим в соответствие значение “фитнесс - функции” – функции, сопоставляющей каждому маршруту число. В оригинальном алгоритме эта функция определяет время прохождения (длину) маршрута. Мы также будем следовать этой идее.

2. Выбирается случайным образом два родителя из популяции и применяем к ним алгоритм скрещивания:

   а) генерируем точку разрыва хромосом родителей

   б) часть первого родителя до точки разрыва копируем в первого потомка

   в) оставшиеся гены второго родителя копируем в первого потомка

   г) оставшиеся гены первого потомка копируем в первого потомка

   д) меняем родителей местами и формируем второго потомка

3. Осуществляется алгоритм мутации:

   а) выбираем пороговое значение и генерируем число в интервале от 0 до 10. Если полученное число выше порога, то:

б) выбираем произвольно два гена из хромосомы и меняем их местами. Проверяем корректность решения.

4. Полученные особи добавляются в популяцию. При этом особи с самым плохим показателем фитнесс-функции удаляются из популяции.
5. Данный процесс повторяется до тех пор, пока не будет выполнено условие остановки. Условия остановки могут формулироваться по-разному. Например:

   а) Остановка после создания k-ого поколения. (k – фиксировано)

   б) Остановка при получении результата с удовлетворяющим нас значением фитнесс-функции.

   в) Остановка при отсутствии изменений между поколениями

Выберем критерий остановки по количеству пройденных поколений

3.4.4 Модифицированный под исходные условия алгоритм

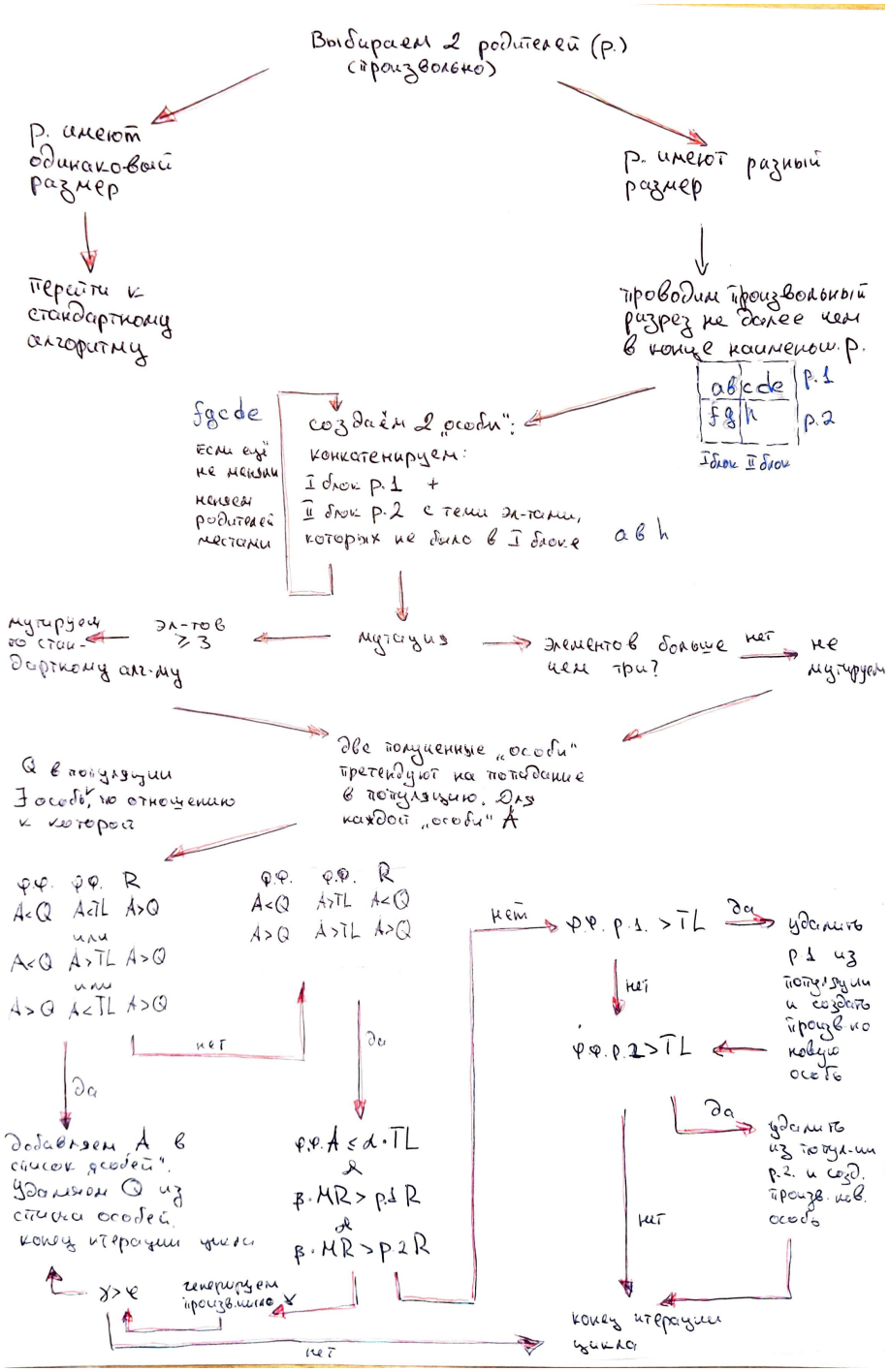
Особенности условия, из-за которых требуется скорректировать работу алгоритма:

1. Городам присваивается число – рейтинг. Это приведет к тому, что надо не слепо минимизировать фитнесс-функцию (время пути) но еще и требовать достижения максимального суммарного рейтинга
2. Накладывается ограничение на время обхода городов (S, далее - TimeLimit). Из этого следует, что поиск гамильтонова цикла в стандартном генетическом алгоритме – лишь частный случай, который вовсе может и не являться решением (к примеру, если время передвижения комивояжера из отправной точки до любого из городов больше, чем заданный TimeLimit, то единственное решение – это остаться в отправной точке
3. Появляется необходимость определить процесс скрещивания для хромосом разной длины
4. Ставится вопрос остановки алгоритма
5. Ставится вопрос о добавлении нового пути

Модифицированный алгоритм удобно изобразить на блок схеме. Предупредим ее обозначениями:

1. p.1 и p.2 - родители 1 и 2 соответственно
2. A – один из двух дочерних маршрутов на этапе скрещивания
3. Q – один из маршрутов текущей популяции
4. P – число элементов в популяции
5. ф.ф - значение фитнесс - функции какого-то элемента
6. Rating (R) – рейтинг маршрута

7. TimeLimit (TL) – ограничение по времени на обход городов
8. MaxRating (MR) = 0 – максимальный рейтинг среди путей, которые удовлетворяют условию TimeLimit
9. GenAmount – число поколений
10. γ – произвольное число от 0 до 100
11. ϕ – пороговое значение (выбираем по усмотрению) - число от 0 до 100
12. α, β – параметры алгоритма - числа от 0 до 1



3.4.5 Обоснование

1. Алгоритм строит неубывающую последовательность “хороших” решений. За счет того, что в популяции всегда сохраняются лучшие маршруты (их процент регулируется параметром alfa от 0 до 1), переход к следующему поколению не сможет ухудшить результат
2. Три основные случая, при которых происходит гарантированное развитие популяции представлены на блок-схеме (а). Также существует три случая, при которых развития популяции при данных новых особях гарантированно не произойдет. Однако ограничиться этими двумя противоположностями несколько бы противоречило вероятностной парадигме алгоритма. Поэтому в рассмотрение вводятся коэффициенты ϕ, γ, которые при удачном подборе могут ускорить работу в спорных вариантах. Вопрос подбора остается на совести статистов.
3. О скорости работы можно сказать следующее: при переходе между поколениями необходимо затратить время на:

   А) скрещивание и мутацию – не более O(n), а в среднем меньше

   Б) пересчет длин маршрутов и рейтинга – не более O(n)

   В) цикл по элементам популяции и сравнение их с потомками – O(число элементов в популяции) « O(n)

Итого переход между поколениями – O(n) Далее необходимо обратиться к статистической литературе, чтобы узнать необходимое число поколений для получения результатов определенной точности. Именно под данную модификацию найти ничего не удалось, однако авторы попробуют сослаться на стандартный генетический алгоритм, по которому существует следующий набор статистических данных [8]:

1. 15 городов.
   Размер популяции 1000
   Результаты:
   число итераций до момента остановки изменений в популиции: 1200,
   время выполнения: 0.321 с,
   погрешность от оптимального (если возможно посчитать): <5%.

2. 40 городов.
   Размер популяции 10000
   Результаты:
   число итераций до момента остановки изменений в популиции: 143000 / 375000,
   время выполнения: 4.3 с / 9 с.

3. 40 городов.
   Размер популяции 10000
   Результаты:
   число итераций до момента остановки изменений в популиции: 101000,
   время выполнения: 3.7 с.

   Последние два примера: первое минимально найденное значение с погрешностью между ними в 3 % было найдено менее чем за 5 сек. Последующие решения отличаются погрешностью меньше 1 %. При правильном задании размеров начальной популяции и процента мутации можно ускорить работу алгоритма [8].

3.4.6 Пример работы алгоритма

В качестве матрицы длин дорог возьмем матрицу 5 на 5

$$\begin{bmatrix} 0 & 14 & 36 & 34 & 22 \\ 14 & 0 & 19 & 26 & 30 \\ 36 & 19 & 0 & 15 & 36 \\ 34 & 26 & 15 & 0 & 21 \\ 22 & 30 & 36 & 21 & 0 \end{bmatrix}$$ Матрица - времени

Массив рейтинга - $[75 \quad 35 \quad 45 \quad 5 \quad 25]$
$TimeLimit = 70$ - ограничение по времени
Размер популяции: 10 маршрутов
Поколение, на котором заканчивается алгоритм: 10

Один из запусков алгоритма выдает следующий результат:
Начальная популяция:

```
Initial population:
GNOME    FITNESS VALUE     RATING
03124          115          185
02134          102          185
0421           77           180
0214           85           180
0432           58           150
0243           93           150
0431           69           140
023            51           125
04             22           100
03             34           80
```

Поколения, на которых произошли положительные изменения результата:

```
Generation 1
Best Result 0432
Max Rating 150
Fitness 58
Time Limit 70
GNOME    FITNESS VALUE    RATING
03124          115          185
02134          102          185
0421           77           180
0214           85           180
0432           58           150
0243           93           150
0431           69           140
0342           91           150
04             22           100
04             22           100
```

```
Generation 2
Best Result 0124
Max Rating 180
Fitness 69
Time Limit 70
GNOME    FITNESS VALUE    RATING
03124          115          185
02134          102          185
0421           77           180
0214           85           180
0124           69           180
0243           93           150
0342           91           150
0431           69           140
04             22           100
04             22           100
```

```
Generation 10
Best Result 0124
Max Rating 180
Fitness 69
Time Limit 70
GNOME    FITNESS VALUE    RATING
03142          126          185
02134          102          185
0421           77           180
0214           85           180
0124           69           180
0124           69           180
0124           69           180
0124           69           180
042            58           145
042            58           145
```

Результат:

Маршрут 0-1-2-4 с максимальным рейтингом 180

4 Сравнение алгоритмов

1) Среди четырех вышеизложенных алгоритмов только один претендует на гарантированное нахождение точного оптимального решения - алгоритм полного перебора. Тем не менее, использование его на практике крайне нежелательно. Действительно, время его работы - $T * n!$, где n - число городов, а T - время формирования одного из вариантов. (Данная оценка напрямую следует из оценки числа перестановок коллекции из n элементов). Более точные результаты см. далее

2) Остальные три алгоритма носят сугубо вероятностный характер. Более того, уверенно утверждать, что один алгоритм работает лучше или хуже другого мешают параметры α, β в муравьином алгоритма, параметры $\alpha, \beta, \gamma, \phi$ и размер популяции в генетическом алгоритме, случайная вероятность p_{rand} в алгоритме иммитации отжига. Именно от правильности подбора этих параметров зависит скорость нахождения хорошего результата.

3) Из результатов тестовых примеров видно, что алгоритм иммитации отжига стправился с задачей быстрее и лучше. Авторы работы, воодушевленные решением задачи, решили протестировать алгоритмы на более интересном наборе данных. И вот что у них получилось.

В качестве временной матрицы была взята матрица 10 на 10:

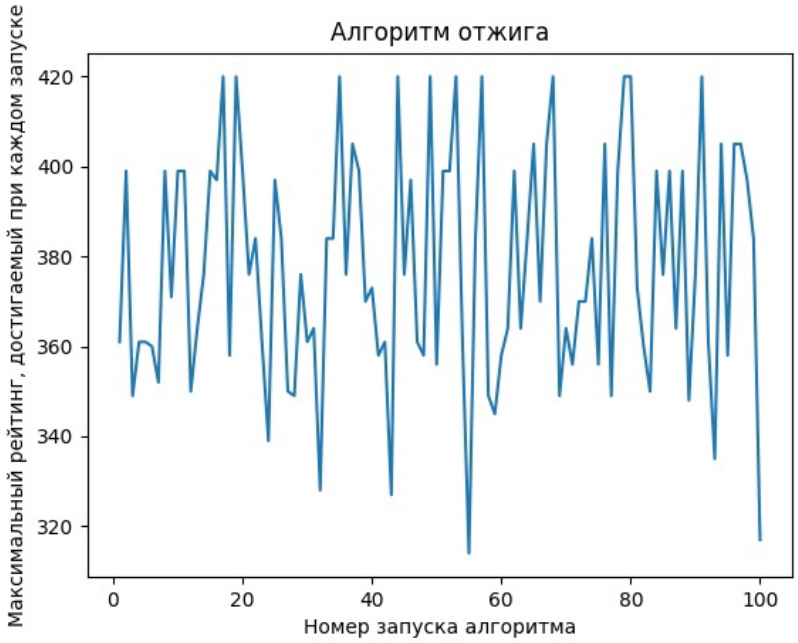
```
40 3 21 70 86 41 99 70 71 71
21 80 88 68 67 86 78 30 84 58
36 80 49 95 8 4 67 70 64 51
2 18 83 13 42 74 94 89 92 19
4 66 61 41 9 3 83 94 91 4
46 7 9 97 76 16 40 38 72 26
82 76 32 1 69 61 9 39 36 23
88 53 3 17 26 44 54 0 39 73
95 38 3 24 48 44 57 71 56 57
17 98 79 75 94 16 55 42 5 30
```

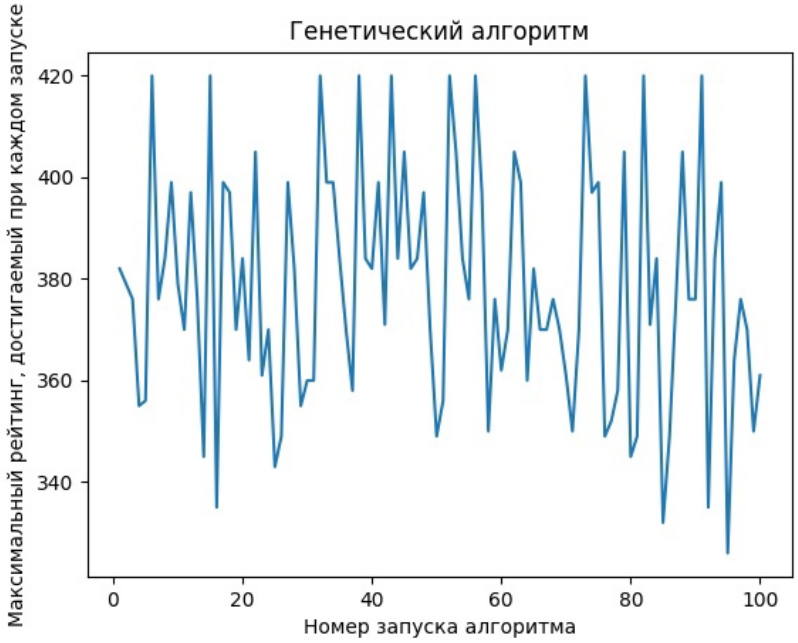
Массив рейтинга - [56 15 47 49 41 23 21 60 50 58]

$TimeLimit = 150$ - ограничение по времени

Запускаются 100 раз два алгоритма: алгоритм иммитации отжига, генетический алгоритм. Каждадая из 100 попыток останавливается на достигнутом по прошествии 150 миллисекунд. Результат фиксируется.

Результаты опыта:





Оба алгоритма дают примерно одинаковый вероятностный показатель в 10% для получения гарантированно наилучшего решения (рейтинг 420 достигается только при обходе каждой из вершин). Однако зачастую результат получается просто сильно приближенным к идеальному. Это, в противовес категоричным результатам тестовых примеров, дает право утверждать, что оба алгоритма достаточно хорошо находят решение. Здесь же хочется дополнить эксперемент оценкой времени работы алгоритма полного перебора. Оказывается, что в среднем алгоритм на тех же данных работает в среднем 8000 миллисекунд, что значительно превосходит 150 миллисекунд вероятностных алгоритмов уже на матрице 10 на 10.

5 Выводы

В нескольких страницах удалось рассмотреть 4 алгоритма решения задачи комивояжера: муравьиный алгоритм, алгоритм иммитации отжига, генетический алгоритм и алгоритм полного перебора. Проделанная работа сформировала несколько важных утверждений относительно решения задачи комивояжера. Во-первых, необходимо понимать, что нахождение гарантированно лучшего решения зачастую требует мощностей, превосходящих мощности имеющиеся. И оттого следует прибегнуть к эврестическим, вероятностным алгоритмам. Во-вторых, среди уже вероятностных алгоритмов может быть достаточно трудно выбрать те алгоритмы, которые однозначно превосходят другие или уступают другим. И, в-третьих, это наталкивает на мысль, что в каких-то случаях было бы весьма разумно обращаться к композиции алгоритмов, находя тем самым некоторый оптимум по времени и энергозатратности.

6 Библиографический список

1. Володина Е.В. Практическое применение алгоритма решения задачи коммивояжера/ Е.В. Володина, Е.А. Студентов
2. Громкович Ю. Алгоритмизация труднорешаемых задач. Часть I. Простые примеры и простые эвристики / Ю. Громкович, Б.Ф.Мельников
3. Громкович Ю. Алгоритмизация труднорешаемых задач. Часть II. Более сложные эвристики. / Ю. Громкович, Б.Ф.Мельников
4. Гудман С. Введение в разработку и анализ алгоритмов: учебное пособие / С. Гудман, С. Хидетниеми
5. Дулькейт В.И. Приближённое решение задачи коммивояжера методов рекурсивного построения вспомогательной кривой
6. Муравьиный алгоритм `https://www.researchgate.net/publication/220203867_Ant_Algorithms_Theory_and_Applications`
7. Муравьиный алгоритм(ч.2) `https://habr.com/ru/post/314056/`
8. Моров В.А. Применение генетического алгоритма к задачам оптимизации `https://vestnik.amursu.ru/wp-content/uploads/2017/12/N57_4.pdf`