Министерство образования и науки Российской Федерации

МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(государственный университет)

ФАКУЛЬТЕТ УПРАВЛЕНИЯ И ПРИКЛАДНОЙ МАТЕМАТИКИ

КАФЕДРА ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Направление подготовки: 03.04.01 Прикладные математика и физика
(магистратура)

“ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ОПТИМИЗАЦИИ ПОДВЯЗКИ ЛОКОМОТИВА И ГЕНЕРАЦИИ РАСПИСАНИЯ ДВИЖЕНИЯ ПОЕЗДОВ”

Выпускная квалификационная работа
(магистерская диссертация)

Выполнил:

студент 273 группы _____Лазарев А.С.

Научный руководитель:

к.ф.-м.н. _____ Коваленко О.В.

Москва 2018

Аннотация

Квалификационная работа на 47 страницах, содержит 16 рисунков, 6 таблиц, использовано 25 источников.

Ключевые слова: генетический алгоритм, методы математического программирования, Java, маршрутизация и планирование маршрута, проблема составления расписания, проблема назначения локомотива.

В данной работе выполнено исследование в области генетических алгоритмов, результатом которого является программный прототип решения. Цель данной работы – проанализировать применение генетических алгоритмов в задачах условной оптимизации и поиска решения в рамках составления расписания движения поездов. Общая задача, поставленная в этой работе, может быть разделена на две подзадачи: анализ принципов работы генетических алгоритмов, их практическое применение в железнодорожном сообщении и разработка программного прототипа решения с применением этого алгоритма. В работе приведены и проанализированы результаты работы этого алгоритма, были даны заключение и рекомендации о возможных использованиях и дальнейших целях разработки и улучшения данного алгоритма.

Оглавление

Введение 4

1. Теоретическая часть. Эволюционные алгоритмы 6

1.1. Общее понятие 6

1.2. Описание генетического алгоритма 7

1.2.1. Общее описание алгоритма 7
1.2.2. Псевдокод 10
1.2.3. Графическая схема 11
1.2.4. Математическое описание генетического алгоритма 12
1.3. Анализ работы эволюционного алгоритма 15

1.4. Выводы о целесообразности применения генетических алгоритмов 18

2. Задача в терминах генетических алгоритмов 21

2.1. Применение генетического алгоритма для решения проблемы расписания поездов 21

3. Практическая часть. Программная реализация задачи 22

3.1. Технические характеристики оборудования. Метод реализации 22

3.2. Представление задачи в терминах генетического алгоритма 23

3.2.1. Представление пространства поиска решения 23
3.2.2. Определение особей: кодирование генотипа 26
3.2.3. Примечание к работе генетических операторов 28
3.2.4. Оператор скрещивания 29
3.2.5. Оператор мутации 31
3.2.6. Оператор селекции, фитнес функция и критерий останова 32
3.2.7. Краткое описание работы алгоритма 33
3.3. Демонстрация работы алгоритма 35

3.3.1. Подготовка данных 35
3.3.2. Введение данных 37
3.3.3. Сравнение результатов 38
3.3.4. Выводы 43
Заключение 44

Список литературы 46

Введение

Генетический алгоритм — это класс адаптивных стохастических алгоритмов с эвристическим поиском и оптимизацией, используемый для решения очень сложных реальных задач, путем моделирования случайного подбора, сочетания и вариации искомых решений с применением последовательности действий, отражающих механизмы природной эволюции[6].

Генетические алгоритмы были впервые использованы [21] Джоном Холландом в 1975 году.

Отличительными чертами алгоритма являются скрещивание, мутация и селекция. Генетический алгоритм работает с данными, чья кодировка обусловлена определением функций скрещивания и мутации. Совокупность всех параметров (генов) одного индивидуума популяции называется генотипом или хромосомой. Селекция – этап, на котором отбирают оптимальных особей для произведения последующих потомств. Обычно, их определяют, основываясь на заданной функции приспособленности (фитнес функции), вычисленной для конкретной особи. Скрещивание – операция обмена генами между двумя лучшими особями. Мутация – операция замены произвольного числа элементов на другие в хромосоме. Как правило, вероятность изменения в хромосоме путем мутации является крайне малой, поскольку процесс мутации позволяет избежать попадания популяции в один локальный максимум[17].

Одной из целей этой работы является исследование генетических алгоритмов, анализ их реализации, а также анализ эффективности их работы.

Основная же цель – это реализации программного прототипа решения. Известно, что задача оптимизации расписания поездов NP-трудная по отношению к числу конфликтов в расписании. Это затрудняет получение хороших решений из-за неразумного вычислительного времени, а также повышает потребность в эвристических методах планирования.

Задача диспетчеризации локомотивов на железных дорогах, рассматриваемая в этой работе, подразумевает оптимизацию поездов на

железнодорожной линии, которая может быть занята другими поездами. Расписание для поездов получено с помощью генетического алгоритма, который включает в себя управляемый процесс для создания начального пула поездов. Каждая особь популяции будет рассматриваться как номинальный объект, написанный на языке Java. Отсутствие примитивного кодирования в двоичный код позволяет применять более сложные операторы мутации и скрещивания. При этом, размер популяции будет ограничен 50 особями. Сочетание таких параметров позволяет алгоритму найти хорошее решение за меньшее время. Результирующим значением фитнес функции генетического алгоритма будет время, пройденное всеми поездами от своих станций отправления и станций прибытия.

1. Теоретическая часть. Эволюционные алгоритмы

1.1. Общее понятие об эволюционном алгоритме

В настоящее время по мере развития информационных технологий и их проникновением во все большее и большее количество сфер и аспектов жизни, возникает все большее количество нерешенных проблем и задач. В частности, возникают задачи поиска решения с разнообразными ограничениями и критериями поиска. Для решения таких задач широко используются эвристические алгоритмы[13]. Эвристические алгоритмы, это те алгоритмы решения задач, которые задействуют практический метод, которых идет по пути повторения естественных процессов, происходящих в природе, при этом не предоставляющий гарантированно точное и/или оптимальное решение, но достаточное для решения задачи. Одним из таких процессов является теория эволюции Дарвина.

Механизмы естественного отбора и эволюции уже изучены, поэтому в настоящее время существует множество примеров алгоритмизации процессов, действующих по принципу естественной эволюции. Алгоритмы, которые входят в эту область, носят название эволюционных алгоритмов. За последние десятилетия было разработано большое количество вариаций таких алгоритмов[15]. Их объединяет одна общая основная идея: входными данными служит популяция особей, которая затем подвергается определенному внешнему воздействию, заставляющее этих особей воспроизводить более приспособленных индивидуумов. В итоге улучшается приспособленность всего поколения, что, в конечном счете, приводит к получению особей, удовлетворяющих заданному целевому условию.

1.2. Описание генетического алгоритма

1.2.1. Общее описание алгоритма

Первым шагом к запуску генетического алгоритма является кодирование пространства поиска. Известно, что кодирование влияет на работу самого алгоритма. Способ кодирования необходимо выбирать исходя из специфики решаемой задачи. Ниже приведены самые частые типы кодирования [11, 23-25]:

а) Двоичный тип. Выбирается тогда, когда признак особи в популяции является двоичным;

б) Численный в бинарном представлении. Является расширенным предыдущим типом. Данный тип является часто применяемым ввиду его просты в реализации;

в) Представление в виде кода Грея. Одним из недостатков такого типа кодирования являются дополнительные затраты вычислительных ресурсов на кодирование и декодирование особи в популяции;

г) Объектные типы.

Затем, задав целевую функцию, к которой стремимся, случайным образом задаем начальную популяцию особей. Основным критерием к начальной популяции является достаточное разнообразие. Это необходимо, чтобы заданная популяция очень быстро не выродилась. Затем, применяем к особям в начальной популяции функцию, результат которой и будет засчитан за значение целевой фитнес функции данной особи. Чем это значение ближе к общему целевому значению, тем лучше кандидат. На основе данной фитнес функции выбираются кандидаты для воспроизведения следующего поколения особей. При этом, чем лучше значение функции, тем больше шанс того, что кандидат даст потомство.

Существует две основных операции, которые влияют на производство потомства:

1) Скрещивание или рекомбинация;

2) Мутация.

Рекомбинацией называется оператор скрещивания двух особей для производства потомства со свойствами, присущими обоим предкам.

На рисунке 1 наглядно показана работа оператора скрещивания двух особей.

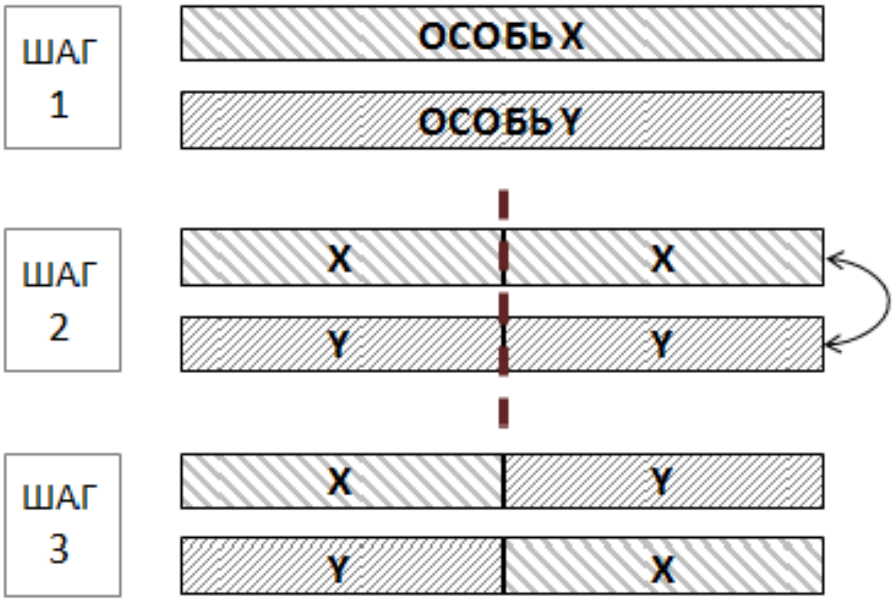

Рис.1 Схема работы оператора скрещивания

Мутация – оператор, который осуществляет механизм внесения случайных изменений в виде новой информации в особь популяции. Он применяется к одному индивидууму, на выходе так же один индивидуум.

В конечном результате, после прохождения одного или двух этапов получается новое поколение особей, с теоретически большим средним значением фитнес функции, чем прошлое. Этот цикл будет продолжаться до тех пор, пока заданное значение в фитнес функции не будет достигнуто, либо пока не будет достигнут предел вычисления по времени или по ресурсам вычислительной системы. Эволюционные алгоритмы часто называют адаптационными, поскольку популяция с каждой итерацией алгоритма имеет свойство адаптироваться под искомое решение.

Заметим, что все процессы, которые происходят во время работы эволюционных алгоритмов, являются стохастическими [21, 22]. Это значит, что более подходящие, по определению фитнес функции, особи имеют больший

шанс дать потомство. Но это не отрицает шанс очень слабых кандидатов (в зависимости от настроек алгоритма) также породить новое поколение. Стоит отметить, что процесс мутации происходит с лишь небольшой долей вероятности. И при этом, заведомо неизвестно, к чему приведет мутация, на каких особях она будет произведена, и в какую ветвь будет развиваться алгоритм далее.

1.2.2. Псевдокод

Ниже приведен псевдокод типичного эволюционного алгоритма:

```
НАЧАЛО
    ИНИЦИАЛИЗАЦИЯ популяции случайно-заданными особями;
    ВЫЧИСЛЕНИЕ значения фитнес функции каждой особи;
    ПОВТОРЯТЬ ПОКА(!условие останова)
        ОТБОР особей-родителей;
        СКРЕЩИВАНИЕ особей-родителей;
        МУТАЦИЯ полученной особи;
        ВЫЧИСЛЕНИЕ значения фитнес функции каждой особи;
        ОТБОР особей-родителей для формирования нового поколения;
    КОНЕЦ ПОВТОРЕНИЯ
КОНЕЦ
```

1.2.3. Графическая схема

Ниже приведена графическая схема типичного эволюционного алгоритма:

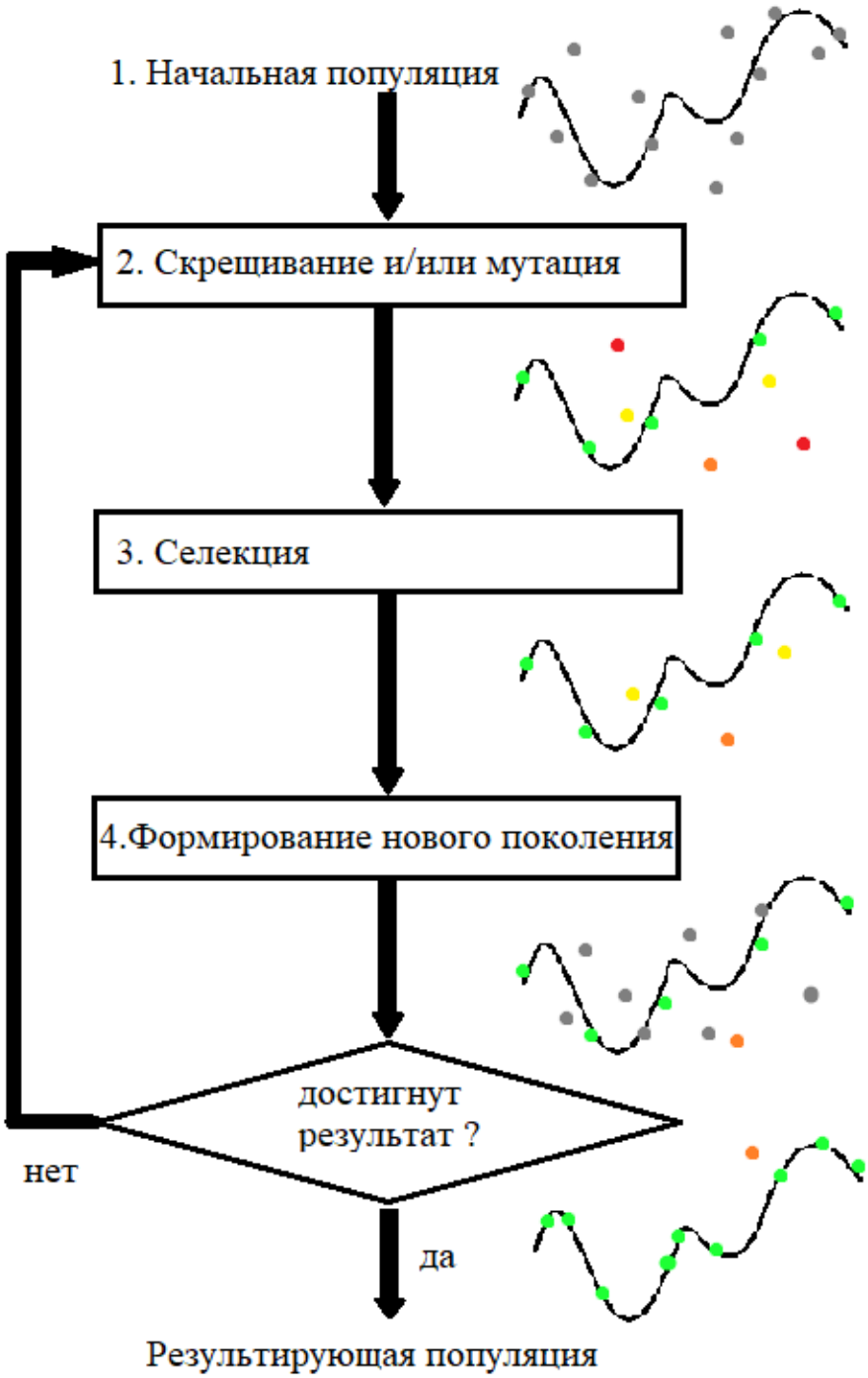

Рис. 2. Схема работы эволюционного алгоритма

На рисунке 2 особи представлены разными цветами: серый – функция пригодности еще не подсчитана; зеленый, желтый, оранжевый и красный цвета означают качество особи, характеризующейся функцией приспособленности.

1.2.4. Математическое описание генетического алгоритма

Сформулируем постановку задачи:

Для некоторой функции приспособленности (фитнес функции) $\boldsymbol{W(x)}$ в пространстве поиска $\boldsymbol{X}$ необходимо найти решение вида (1) [2, 3, 11, 14]

$$\boldsymbol{x}^* = \mathbf{argmin}\, \boldsymbol{W(x)}, \text{ где } \boldsymbol{x \epsilon X} \quad (1)$$

или же вида (2)

$$\boldsymbol{x}^* = \mathbf{arg}\boldsymbol{max W(x)}, \text{ где } \boldsymbol{x \epsilon X} \quad (2)$$

в зависимости от типа задачи.

Тогда описание работы алгоритма зададим следующим образом:

1) Произвольным образом задается конечный набор (популяции) начальных решений (3):

$$\boldsymbol{P^1 = \{p_1^1 \dots p_n^1\}}, \quad (3)$$

где $\boldsymbol{p_i^1 \epsilon X}$ - первое поколение особей, а $\boldsymbol{n}$ - размер всей популяции в системе.

2) Производится оценка приспособленности текущего $\boldsymbol{k}$ решения (текущей популяции) (4):

$$\boldsymbol{F^k = \{f_1^k \dots f_n^k\}}, \text{ где } \boldsymbol{f_i^k = W(p_i^k)}. \quad (4)$$

3) Выход из итераций алгоритма, при срабатывании условия останова, иначе:

4) Происходит генерация нового набора решений (новой популяции) с использованием операторов селекции $\boldsymbol{S}$, мутации $\boldsymbol{M}$ и скрещивания $\boldsymbol{C}$ в виде (5) и осуществляется переход на 2 шаг:

$$\boldsymbol{P^{k+1} = C \cdot M \cdot S(P^k, F^k)} \quad (5)$$

Оценка приспособленности:

Часто такая оценка производится не в одну, а в две стадии. Первая стадия – непосредственно сама оценка (6):

$$\boldsymbol{F^k = \{f_1^k \dots f_n^k\}}, \text{ где } \boldsymbol{f_i^k = W(p_i^k)}. \quad (6)$$

Вторая стадия – необходимые преобразования, например, нормировка в виде (7):

$$\boldsymbol{F^{k\prime}} = \{\boldsymbol{f_1^{k\prime}} \dots \boldsymbol{f_n^{k\prime}}\}, \tag{7}$$

где $\boldsymbol{f_i^{k\prime}} = (\boldsymbol{f_i^k} - \boldsymbol{f_0})/(\boldsymbol{f_1} - \boldsymbol{f_0})$,

$\boldsymbol{f_0}$ и $\boldsymbol{f_1}$ – самый плохой и самый хороший показатели в текущей популяции.

Оператор скрещивания

При задании особей в одноточечном виде, результатом скрещивания двух особей, заданных в виде (8)

$$\boldsymbol{p_i^k} = \{\boldsymbol{a_1}, \boldsymbol{a_2}, \dots, \boldsymbol{a_n}\} \text{ и } \boldsymbol{p_j^k} = \{\boldsymbol{b_1}, \boldsymbol{b_2}, \dots, \boldsymbol{b_n}\} \in \boldsymbol{P^k} \tag{8}$$

в $\boldsymbol{k}$-том наборе решений (популяции) станут два элемента набора решений $\boldsymbol{k+1}$, такие, что (9, 10):

$$\boldsymbol{p_i^{k+1}} = \{\boldsymbol{a_1}, \dots, \boldsymbol{a_m}, \boldsymbol{b_{m+1}}, \dots, \boldsymbol{b_n}\} \tag{9}$$

и

$$\boldsymbol{p_j^{k+1}} = \{\boldsymbol{b_1}, \dots, \boldsymbol{b_m}, \boldsymbol{a_{m+1}}, \dots, \boldsymbol{a_n}\} \in \boldsymbol{P^{k+1}} \tag{10}$$

В (9, 10) точка $\boldsymbol{m}$ выбирается произвольно. При задании в двухточечном виде, случайных точек пересечения в генотипах родительских особей будет также две. Таким образом, этот оператор также применим и для $\boldsymbol{n}$-точечного вида.

Оператор мутаций

Оператор мутаций заменяет произвольное число генов у особи на другие случайные гены. Например, имеется особь (11)

$$\boldsymbol{p_i^k} = \{\boldsymbol{a_1}, \dots, \boldsymbol{a_o} \dots, \boldsymbol{a_p} \dots, \boldsymbol{a_q}, \dots, \boldsymbol{a_n}\}, \tag{11}$$

то при воздействии на неё оператором мутации $\boldsymbol{M}$ получим (12):

$$\boldsymbol{M} \cdot \boldsymbol{p_i^k} = \{\boldsymbol{a_1}, \dots, \boldsymbol{a_o'}, \dots, \boldsymbol{a_p'}, \dots, \boldsymbol{a_q'}, \dots, \boldsymbol{a_n}\} \tag{12}$$

Оператор мутаций позволяет вносить новые элементы в популяцию и помогает избежать вырождаемости популяции.

Оператор селекции

Оператор производит отбор тех особей $\boldsymbol{p_i^k}$, чья оценка приспособленности (13) является одной из наилучших.

$$\boldsymbol{f_i^k} = \boldsymbol{W}(\boldsymbol{p_i^k}) \tag{13}$$

1.3. Анализ работы эволюционного алгоритма

Как уже описывалось ранее, для работы эволюционного алгоритма требуется провести несколько шагов:

1) выбрать формат представления особей в популяции;

2) Обозначить фитнес-функцию;

3) Определить параметры формирования популяции;

4) Задать функцию отбора родителей;

5) Определить параметры операторов скрещивания, мутации;

6) Задать функцию определения живучести.

Помимо этого должны быть определены условия старта и остановки алгоритма.

Чтобы продемонстрировать, как работает эволюционный алгоритм и почему он является результативным при решении задач широкого спектра, можно показать алгоритм его работы визуально [3,13,19].

На рис. 3, рис. 4 и рис. 5 изображены три различных по временной шкале этапа в работе эволюционного алгоритма, наглядно показывающих распределение особей в начале работы алгоритма (рис. 3), в середине (рис. 4) и в конце работы алгоритма (рис. 5).

Рис. 3. Распределение индивидуумов в начале работы эволюционного алгоритма

Рис. 4. Распределение индивидуумов в середине работы эволюционного алгоритма

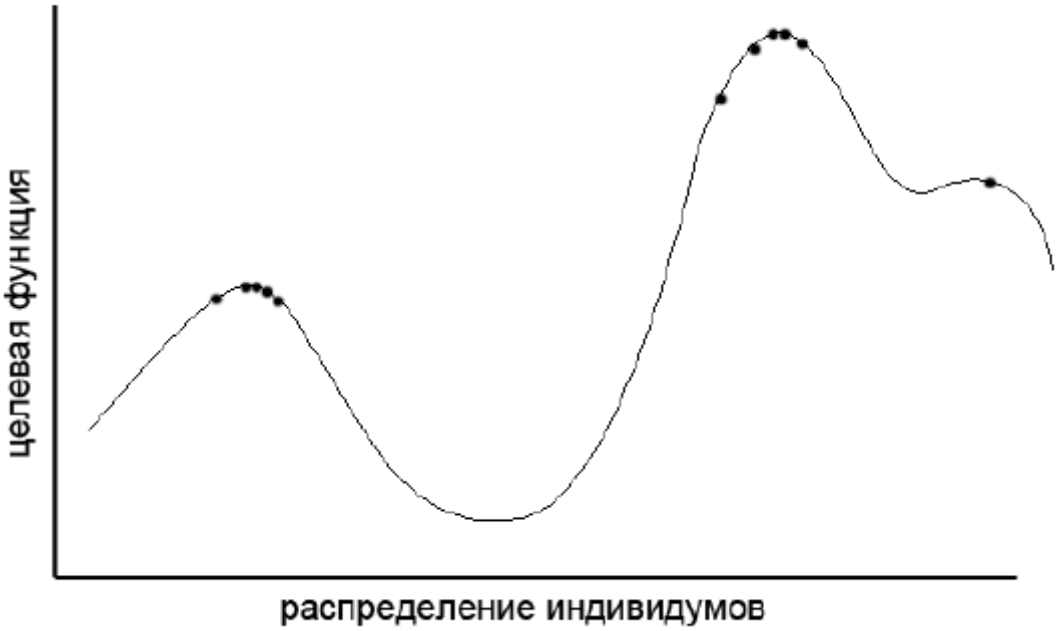

Рис. 5. Распределение индивидуумов в конце работы эволюционного алгоритма

В начальной фазе, сразу после инициализации, хромосомы распределены случайным образом по всей области определения. После нескольких циклов работы алгоритма (селекции, скрещивания и мутации) популяция отбросила самых неприспособленных, согласно оценке фитнес функции, особей и произвела новых потомков из тех, что были ближе к вершине, тем самым улучшая среднее значение функции фитнеса всей популяции.

1.4. Выводы о целесообразности применения генетических алгоритмов

В пункте «1.3 Описание работы генетического алгоритма» данной работы было показано, что по завершению работы алгоритма все особи находятся в локальных максимумах. Стоит помнить, что всегда имеется шанс, что все кандидаты выродятся в локальный максимум, который не будет являться глобальным. [1, 11] Однако, шанс этого события крайне мал. Ведь изначально особи случайным образом распределяются максимально равномерно по всей области определения. Это и не позволяет всем индивидуумам попасть в одну точку, только если эта точка не является единственным максимумом. Для уменьшения шанса попадания всех особей в популяции в локальный максимум, как правило, заранее задается некоторая область на всем пространстве, в пределах которой алгоритм будет искать максимумы.

Еще одним положительным свойством эволюционного алгоритма, которое характерно всем итерационным алгоритмам, оперирующим заранее заданным набором решений, является эффективность в любой точке во времени. [6, 12] Это значит, что алгоритм может начинать работать с любого заданного набора и пытаться его улучшить, действуя бесконечно. Это свойство вызывает проблему нахождения оптимального соотношения между вычислительными ресурсами, требуемыми для дальнейшей работы алгоритма, и временем поиска более хорошего решения. На рисунке 6 приведен график увеличения наилучшего значения фитнес функции в поколении по мере работы алгоритма.

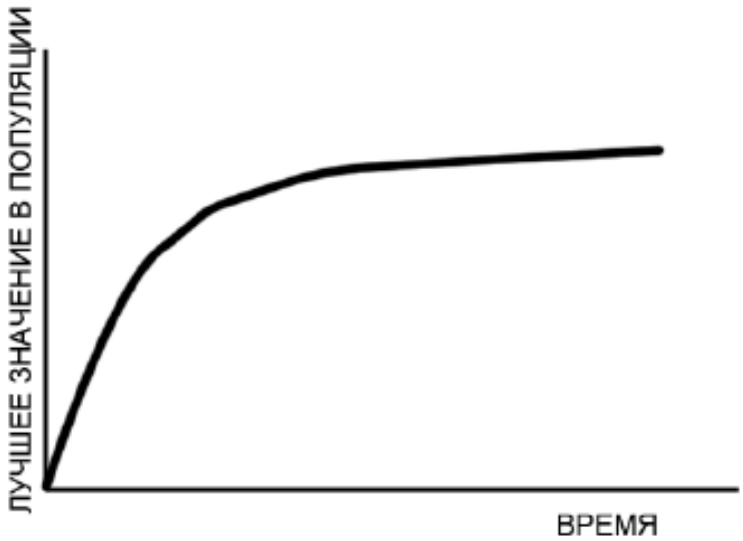

Рис. 6. Улучшение значения фитнес функции каждой особи по мере работы эволюционного алгоритма

На графике, изображенном на рисунке 6 видно, что значение фитнес функции имеет логарифмический рост. Это означает, есть определенное условие останова, при котором приращение значения фитнес функции будет нерентабельно относительно затраченного времени для его вычисления. [3] Эта условие зависит от конкретной решаемой задачи, то есть является субъективным.

Несмотря на критику генетических алгоритмов, можно утверждать, что генетические алгоритмы хорошо применимы в NP-сложных задачах, [1, 5, 6] где найти точное решение не представляется возможным, однако, следует учесть, что для каждой конкретной задачи приходится подбирать свои параметры работы генетического алгоритма.

Подводя рассмотрение эволюционного алгоритма к завершению, стоит также указать на результаты исследования Дэвида Гольдберга [2], который сравнил разные типы алгоритмов на широком спектре задач и привел их эффективность. Усредненные и абстрагированные результаты представлены на рисунке 7.

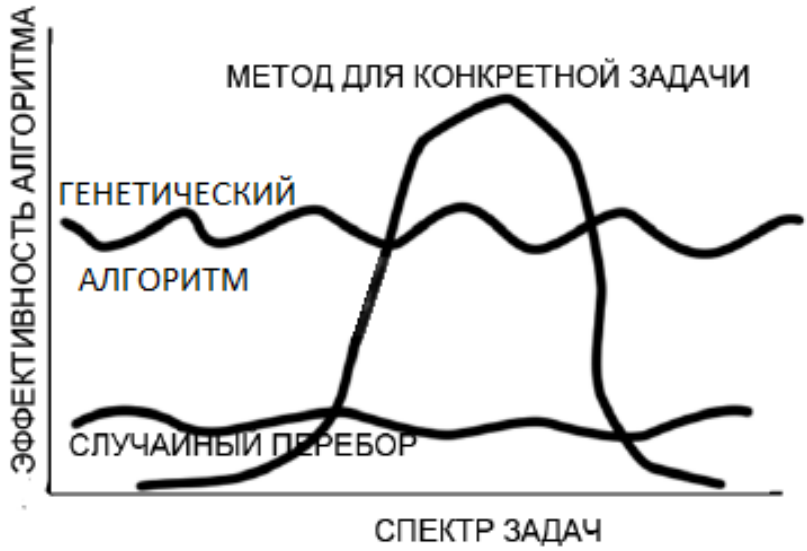

Рис. 7. Сравнение эффективности работы разнообразных типов алгоритмов по Дэвиду Гольдбергу

2. Задача в терминах генетических алгоритмов

2.1. Применение генетического алгоритма для решения проблемы расписания поездов

Генетические алгоритмы успешно применяются к комбинаторным проблемам и способны обрабатывать огромные пространства поиска, которые возникают, например, в задачах планирования в реальной жизни [1, 4 ,7].

Проблема расписания поездов - сложная и трудоемкая задача. Огромное пространство для поиска решения, делает генетические алгоритмы одним из лучших подходов к эффективному решению такой задачи.

В представленной работе рассматривается решение проблемы оптимизации подвязки локомотива и проблема генерации расписания движения поездов. По мере выполнения научно-исследовательской работы стало ясно, что выбранная тема очень обширная, то было приято решение более подробно рассмотреть проблему генерации расписания движения поездов.

При составлении расписания поездов следует учитывать пропускную способность линии, возможные коллизии по расписанию с другими поездами, возможность ввода новых поездов в дополнении к уже готовому расписанию и другие эксплуатационные ограничения. Традиционно расписания движения поездов создавались вручную и настраивались так, чтобы выполнялись все ограничения. [7 - 9] В настоящее время, в рамках сильной конкуренции, высокой скоростью роста железнодорожного сообщения и появлением новых технологий наряду с увеличением скорости работы компьютеров, все вышеперечисленное являются причинами, которые оправдывают необходимость использования автоматических инструментов, позволяющих эффективно создавать приемлемые и оптимизированные расписания.

Предполагая, что проблема составления расписания поездов является очень сложной задачей, а генетический алгоритм - подходящей процедурой для ее решения, я разработал генетический алгоритм для этой задачи.

3. Практическая часть. Программная реализация задачи

3.1. Технические характеристики оборудования. Метод реализации

Во время своей исследовательской деятельности я использовал ноутбук со следующими техническими характеристиками:

- Двуядерный процессор Intel Core i7-4510U, с частотой 2ГГц;
- Оперативная память 8 ГБ класса DDR3;
- 64-разрядная операционная система Windows 10 профессиональная.

Для реализации программного прототипа решения я выбрал объектно-ориентированный язык Java версии 8.

Работа по реализации программного прототипа решения проводилась в свободной интегрированной среде разработки «Eclipse IDE» версии Oxygen.3a Release (4.7.3a).

Написанные классы представлены на рисунке 8.

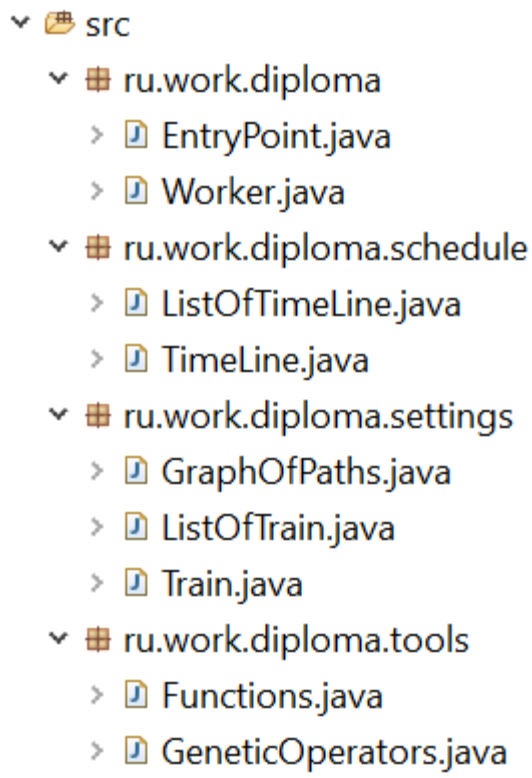

Рис. 8 Структура классов Java

3.2. Представление задачи в терминах генетического алгоритма

3.2.1. Представление пространства поиска решения

Чтобы применить генетический алгоритм к какой-либо конкретной проблеме, необходимо внутреннее представление пространства решений. Правильность этого выбора обуславливает успех или неудачу применения генетического алгоритма к конкретной задаче.

Как было сказано ранее в пункте «1.2.1. Общее описание алгоритма», существует множество способов кодирования. Проанализировав каждый из них, я пришел к выводу, что в данной работе лучше использовать список действий или графиком действий в виде представления хромосомы (вида решения), так как эта кодировка широко используется в задачах планирования [1, 17, 18].

Определим общие входные данные, необходимые для работы программного прототипа решения:

1. Поезда.
   - 1.1. Количество поездов;
   - 1.2. Поезд должен иметь станцию отправления и конечную станцию;
   - 1.3. Поезда также имеют такой атрибут как минимальное время в пути от станции отправления до конечной станции. Выражается в минутах и может быть задано на входе или же рассчитано алгоритмом;
   - 1.4. Поезд может быть приоритетным: известно, что пассажирские поезда, более приоритетные, чем грузовые;
   - 1.5. Поезд может иметь фиксированное время отправления и время прибытия на конечную станцию.
2. Железнодорожная транспортная сеть.
   - 2.1. Количество станций;
   - 2.2. Связи станций. На данном этапе необходимо установить, как каждая из станций связана с другой;

2.3. Необходимо указать, количество путей, которыми связана одна станция с другой;

2.4. Для корректной работы необходима также информация о том, сколько в минутах при определенной средней скорости занимает перемещение по каждому из путей от станции к станции.

В реализованном программном прототипе решения поезда представлены в виде объектов, описанных классом $Train.java$. Объекты этого класса инициализируются входными значениями.

Железнодорожная транспортная сеть представлена в виде графа смежности, который в свою очередь представлен объектом, описанным в классе $GraphOfPaths.java$. Разберем задание графа матрицы на примере трех станций.

Пусть имеются три станции: станция А, станция Б и станция В. Тогда, создаем матрицу 3х3, на главной диагонали которой будут находится эти станции. Это представлено на рисунке 9.

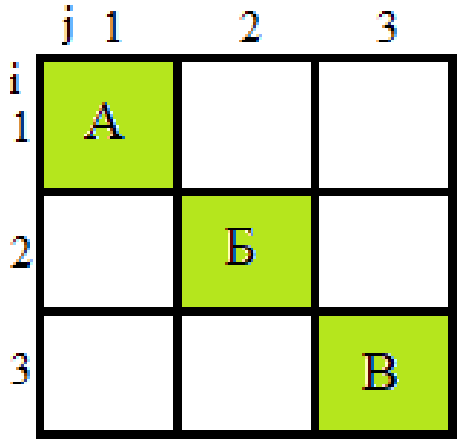

Рис. 9 Представление станций в матричном виде

Тогда, пересечение $\boldsymbol{i}$–той строки и $\boldsymbol{j}$-того столбца, при условии, что $\boldsymbol{i} \neq \boldsymbol{j}$, даст мне возможность задать некоторые параметры, которые относятся к паре станций. Например, таким образом можно задать количество путей между станциями, при $\boldsymbol{i} > 1, \boldsymbol{j} > \mathbf{1}$ и $\boldsymbol{i} \neq \boldsymbol{j}$.

i \ j	1	2	3
1	А	1	0
2		Б	1
3			В

Рис. 10 Описание количества связей между станциями

На рисунке10 видно, что станции А и Б, ровно как и станции Б и В имеют по одному пути, а станции А и В не имеют прямого пути не в обход станции Б. Заметим, что на главной диагонали количество ожидающих поездов, согласно настройке алгоритма, не ограничено, что легко можно поменять. Граф представлен в программного прототипа решения, как переменная $pathGraph$.

Похожим образом можно описать время, затрачиваемое поездами в пути (рисунок 11.)

i \ j	1	2	3
1	А	10	0
2		Б	25
3			В

Рис. 11 Описание времени движения поездов между станциями

Время в пути от станции А до станции Б 10 минут, а от станции Б до станции В 25 минут. Проезда между станциями А и В нет. Граф представлен в программе, как переменная $timeGraph$.

3.2.2. Определение особей: кодирование генотипа

Итак, если генотип является графиком действий, то ген представляет собой часть этого графика. Ген представлен в разработанном программном продукте как объект, описанный в классе $TimeLine.java$, структура которого, в свою очередь, представлена в следующем виде на рисунке 12:

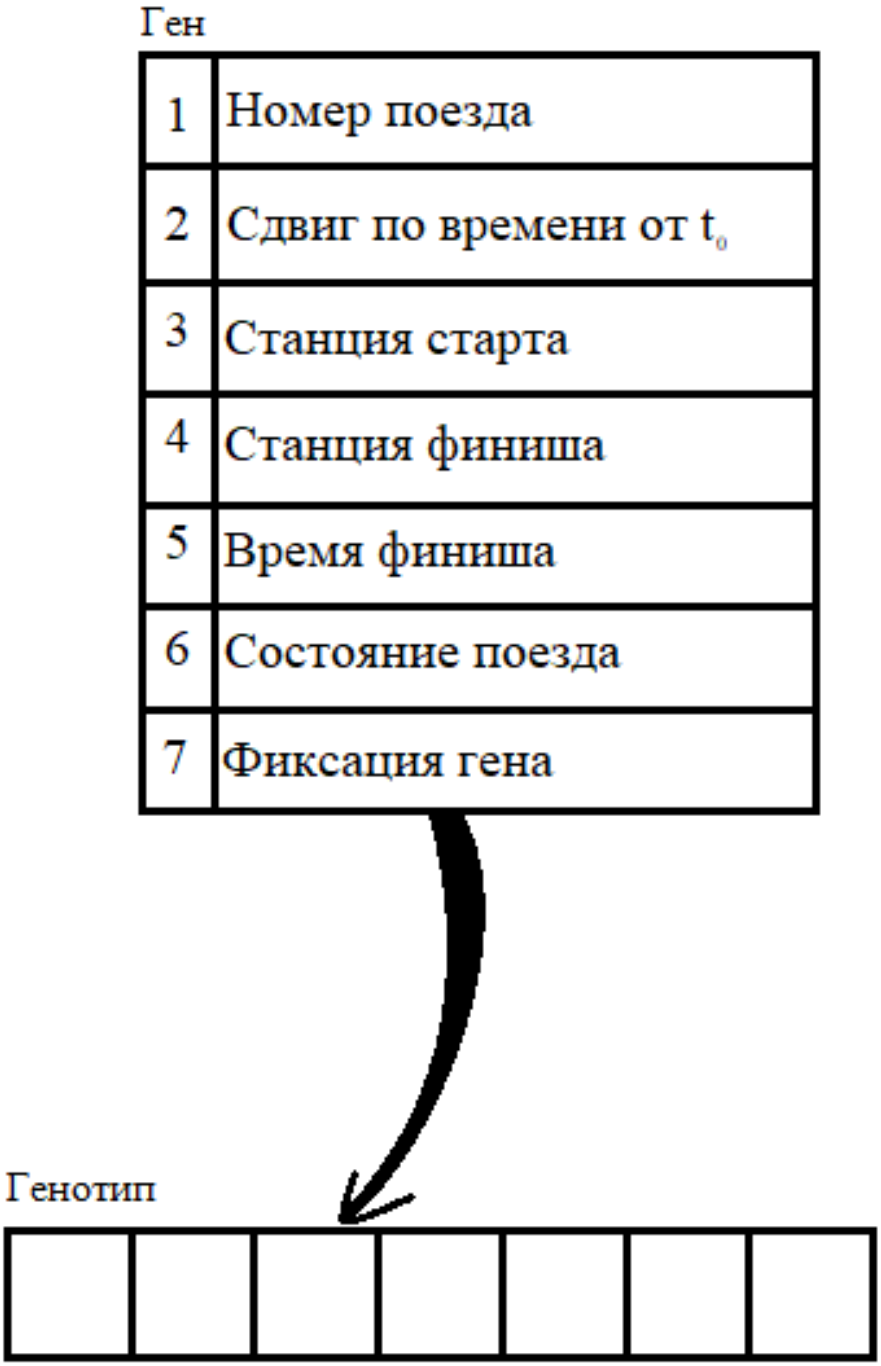

Рис. 12 Представление гена в генотипе

Ген, как частица совокупного графика, представленный на рисунке 12, несет в себе некоторую информацию. Разберем ее более подробно:

а) Номер поезда;

б) Сдвиг по времени от $\boldsymbol{t_0}$ **(tShift)**, где $\boldsymbol{t_0}$ – начальная временная точка запуска алгоритма;

в) Состояние поезда говорит о том, что происходит с поездом в данный момент времени. У него может быть только два состояния: ожидание (waiting) или движение (moving);

г) Станции старта, - несет информацию, с какой точки поезд начал движение в момент времени сдвиг по времени от начального точки $\boldsymbol{t_0}$, заданной в пункте (б);

д) Станция финиша является точкой назначения и может совпадать со станцией старта, если поезд находится в состоянии ожидания;

е) Время финиша **(tEnd)** является предполагаемым и рассчитывается как время, требуемое для перемещения от станции старта (г) до станции финиша (д). В случае, если поезд находится в состоянии ожидания, время финиша является временем ожидания и задается согласно настройкам алгоритма;

ж) Фиксация гена отражает возможность неизменяемости гена в генотипе особи. Это необходимо, для поездов с фиксированным временем отправления и прибытия.

Стоит заметить, что все гены в особи можно сгруппировать по номеру поезда, отсортировав по времени сдвига от $\boldsymbol{t_0}$, что, в конечном счете, и является некоторым решением в виде расписания для каждого конкретного поезда.

Как можно видеть, объектная кодировка является очень гибкой с точки зрения добавления свойств гена и поддержки программного продукта в целом. Выбранная кодировка позволяет легко добавлять или удалять критерии работы генетических операторов, что, в свою очередь, повышает качество решения усложненной таким образом задачи оптимизации.

3.2.3. Примечание к работе генетических операторов

Если в результате работы генетических операторов нарушилась целостность расписания или возникли коллизии в расписании, то применение функции $validateIndivid()$ решит эту проблему. Она достроит хромосому, создав новые гены с ожиданием для поездов, которые конфликтуют или уберет такие блоки, если они лишние. Функция также выполнит проверку на смещение сдвигов по времени в генах, если это необходимо.

3.2.4. Оператор скрещивания

Одним из уникальных и важных механизмов, играющих большую роль в работе, генетических алгоритмов является рекомбинация, выступающая в виде оператора скрещивания (или кроссовера). Этот оператор объединяет набор генов двух родительских хромосом с образованием двух потомков, которые наследуют их характеристики.

Индивидуумы в популяции смешиваются друг с другом случайным образом, и каждую пару обрабатывает оператор скрещивания, создавая двух потомков. Затем родительские особи заменяются потомством. Оператор скрещивания должен сохранять и сочетать хорошие гены блоки для создания более хороших особей. [5]

Поскольку в данной работе используется объектное кодирование генов, то было принято решение о реализации объектного оператора скрещивания.
Оператор скрещивания, как и прочие другие, описан в виде функций в классе $GeneticOperators.java$.

Все гены в особи можно сгруппировать по номеру поезда таким образом, что разработанный генетический оператор сможет скрещивать особи, заменяя одну группу ген на другую. При этом, максимальное число групп дня обмена определяется как количество поездов $\boldsymbol{N-1}$.

Пусть, например, мы имеем три поезда $\boldsymbol{a,b,c}$ и две особи $\boldsymbol{G}$ и $\boldsymbol{H}$, где цветовая окраска гена будет символизировать гены одной группы поездов: $\boldsymbol{a}$ - красный, $\boldsymbol{b}$ - желтый и $\boldsymbol{c}$ – зеленый (рисунок 13).

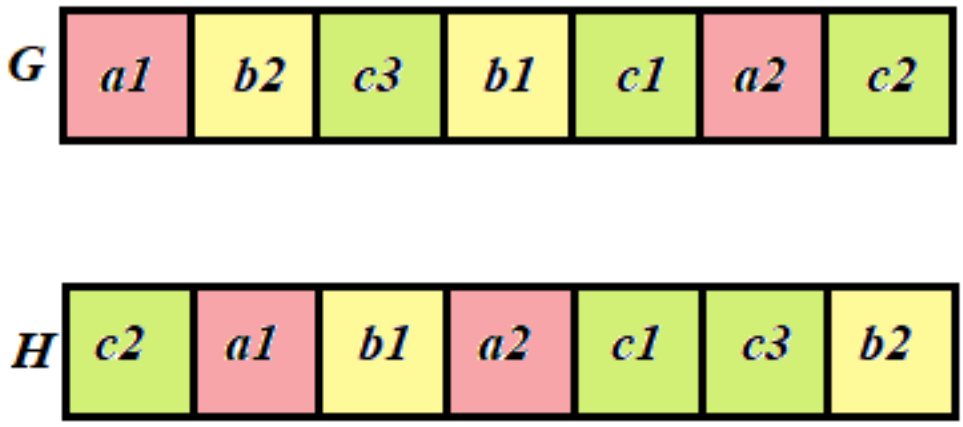

Рис. 13 Представление генов в генотипе

У особей-родителей $\boldsymbol{G}$ и $\boldsymbol{H}$ произошел обмен информацией (генами) о поезде $\boldsymbol{a}$. Тогда, результатом работы оператора скрещивания будут особи-потомки $\boldsymbol{G'}$ и $\boldsymbol{H'}$, изображенные на рисунке 14.

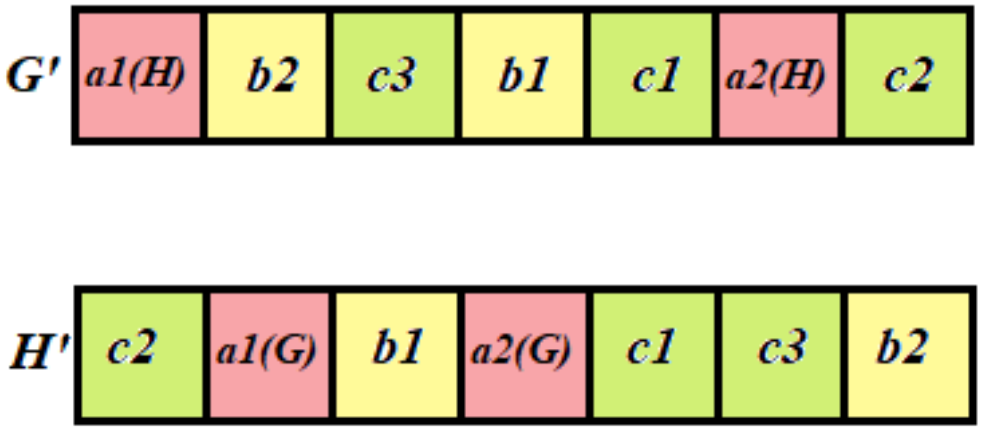

Рис.14 Результат работы оператора скрещивания

На рисунке 14 изображен случай, когда длина генотипов особей равна, и количество генов с информацией о поезде $\boldsymbol{a}$ тоже равно, но такое бывает далеко не всегда.

Стоит заметить, что порядок генов в генотипе ни на что не влияет.

3.2.5. Оператор мутации

После того как оператор скрещивания был применен, и популяция потомков заменила родительскую популяцию особей, применяется оператор мутации. Мутация изменяет один или несколько генов выбранной хромосомы, внося тем самым некоторую дополнительную изменчивость в популяцию.

Реализованный оператор мутации работает следующим образом: для каждой особи выбирается случайная группа генов, отвечающая за расписание одного из поездов. И если срабатывает шанс мутации, который равен $P_M = 0{,}05$, то включается следующий алгоритм. Поскольку улучшение допустимых решений представляет особый интерес, то новая группа должна быть более хорошей по оценке функции фитнеса $calcFitness()$, чем ее предшественница. Функция $genNewPath()$ пытается сгенерировать новый путь от станции отправления поезда до конечной станции. Новый путь заменяет старый в особи.

3.2.6. Оператор селекции, фитнес функция и критерий останова

Оператор селекции - это искусственная версия природного механизма, в ходе которого выживают наиболее приспособленные особи. В природе конкуренция между индивидуумами за ограниченные ресурсы приводит к тому, что наиболее приспособленные индивиды доминируют над более слабыми.

Основываясь на некотором относительном качестве, гены имеют определенное число своих копий в популяции. Более приспособленный индивид получает большее число потомков и, следовательно, имеет более высокую вероятность выживания в последующем поколении.

В данной работе оператор селекции реализован следующим образом. Алгоритм, заложенный в операторе селекции программного продукта, работает с двумя случайно выбранными индивидами, которые «конкурируют за выживание». Для каждого из них происходит оценка по фитнес функции. Тот индивид, который имеет лучшую фитнес-ценность, появится в последующей популяции. Этот цикл повторяется $\boldsymbol{N}$ раз, пока $\boldsymbol{N}$ особей не будут выбраны для следующей популяции, где $\boldsymbol{N}$ – размер популяции.

Фитнес функция представляет собой отклонение от величины субъективного поезда $tTravelMin$, - минимального времени от станции отправления до конечной станции.

Критерий останова в реализованном алгоритме может быть настраиваемым, поскольку он зависит от таких факторов как, размер транспортной сети (числа станций, перегонов), количества поездов.

Критерий останова определяется в виде (14)

$$\sum_{i,j=0}^{m-1}(tEnd_j - tShift_j) - \sum_{i=0}^{n-1} 1{,}5 \cdot (tTravelMin_{\mathrm{i}}) \leq 0 \qquad (14)$$

где n – количество поездов,

m – количество генов в текущей особи для i поезда

3.2.7. Краткое описание работы алгоритма

После запуска программы в $main()$ функции в классе $EntryPoint.java$, создается новый объект типа Worker. У этого объекта вызывается функция $workflow()$, которая, в свою очередь, запускает работу всего алгоритма. Далее пользователю будет предложено ввести вводные данные, речь о которых шла в пункте «3.2.1. Представление пространства поиска решения» данной работы. На данном этапе происходит обработка и проверка всех данных, которые вводит пользователь, и если пользователь введет что-то прямо противоречащее логике работы алгоритма, то в консоли выведется информация об этом. Происходит наполнение объекта Worker, структура которого изображена на рисунке 15.

```
1  package ru.work.diploma;
2
3  import java.util.List;
4
5  import ru.work.diploma.schedule.ListOfTimeLine
6  import ru.work.diploma.settings.GraphOfPaths;
7  import ru.work.diploma.settings.ListOfTrain;
8  import ru.work.diploma.settings.Train;
9  import ru.work.diploma.tools.Functions;
10
11 public class Worker {
12
13     private int popSize = 50;
14     private double mutProb = 0.05;
15     private GraphOfPaths graph;
16     private GraphOfPaths timeGraph;
17     private ListOfTimeLine listOfTimeLine;
18     private ListOfTrain listOfTrain;
19     private List<ListOfTimeLine> population;
```

Рис. 15 Структура класса Worker.java

Объекту Worker принадлежит объект-атрибут $population$ типа List<ListOfTimeLine>. На самом деле это список списков, состоящих из генов,

описанных в пункте «3.2.2. Определение особей: кодирование генотипа». Работы по заполнению объекта $population$ берет на себя функция $generateTimeLine()$.Генерация генотипа происходит для всех поездов с одинаковым стартовым временем. Это позволяет уменьшить общее временное окно занятости железнодорожной сети. Эта функция является итеративной, число итераций popSize. Метод $generateTimeLine()$ проходится по всем поездам и каждый момент времени (15) и проверяет, занят ли каждый из поездов в этот момент, если нет, то создает новый ген $TimeLine$.

$$t_{new} = t_{old} + shift \tag{15}$$

Создание пути движения поездов происходит таким образом, что алгоритм создания нового гена в функции $generateTimeLine()$ проверяет предыдущий ген $TimeLine$, и не пускает поезд в том направлении, откуда он приехал.

После определения начальной популяции подсчитывается пригодность функцией $calcFitness()$ каждой особи, определенного как тип $ListOfTimeLine$. Затем проверяется критерий останова в виде функции $checkForBreak()$. Если он срабатывает, то мы делаем вывод о том, что задача решена случайным методом поиска решения. Если такое не происходит, то запускается генетический алгоритм в блоке $while(!\,checkForBreak())$, более подробно описанный в пункте «1.2.1. Общее описание алгоритма» и проиллюстрированный в пунктах «1.2.2. Псевдокод» и «1.2.3. Графическая схема» главы 1. В данном блоке задействуются генетические операторы, описанные в пункте «3.2.3. Примечание к работе генетических операторов», пункте «3.2.4. Оператор скрещивания», пункте «3.2.5. Оператор мутации» и пункте «3.2.6. Оператор селекции, фитнес функция и критерий останова» данной главы.

3.3. Демонстрация работы алгоритма

3.3.1. Подготовка данных

Пусть имеется 7 абстрактных станций: A, B, C, D, E, F, G. Перечисленные станции произвольно соединены между собой одинарными путями. В третьей колонке таблицы 1 указано, с какими станциями соединяется станция из второй колонки и сколько требуется времени в минутах для преодоления этой дистанции (четвертая колонка таблицы 1).

Таблица 1 Схема железнодорожной сети

№	Станция	Станция соединения	Время, м
0	A	B	20
		C	10
		F	15
1	B	A	20
		C	10
		F	15
2	C	A	10
		B	10
		D	10
3	D	C	10
		E	10
		F	10
4	E	B	15
		D	10
		G	15
5	F	A	15
		D	10
		G	10
6	G	E	15
		F	10

Изобразим таблицу 1 в виде схемы железнодорожной сети на рисунке 16.

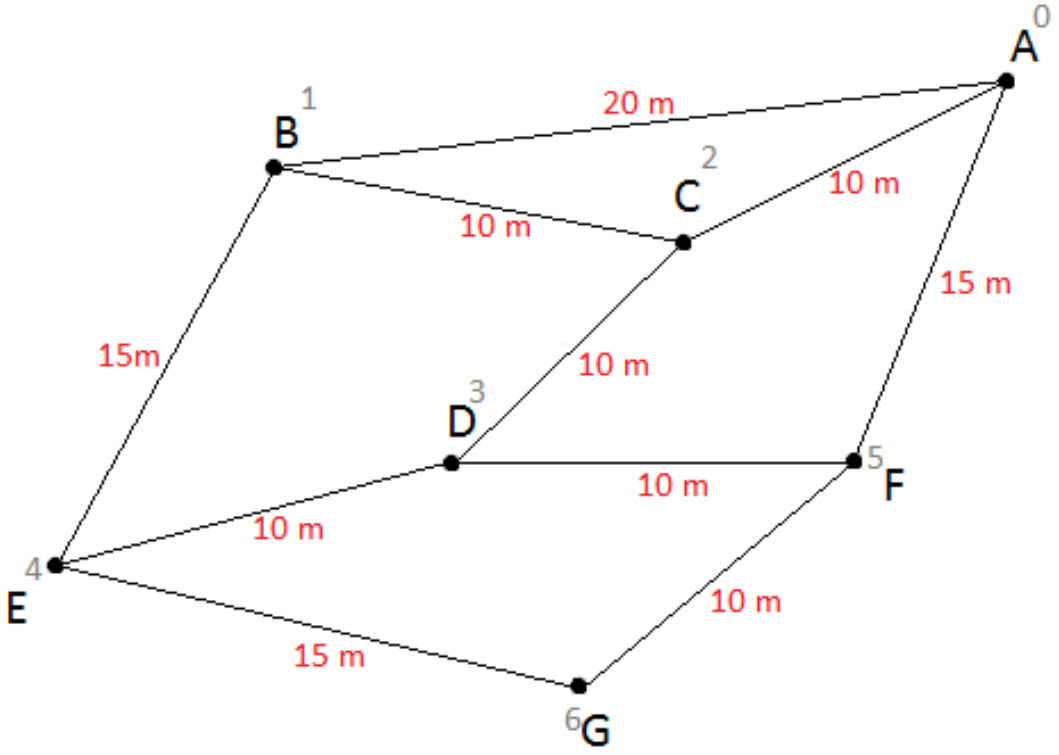

Рис. 16 Схематичное изображение железнодорожной сети

Зададим в виде таблицы 2 курсирующие поезда по железнодорожной сети на рисунке 16.

Таблица 2 Поезда в тестовой железнодорожной сети

№	Станция отправления	Конечная станция	Кратчайший путь по вершинам	Время преодоления, м
1	A	E	A-C, C-D, D-E	30
2	E	A	E-D, D-C, C-A	30
3	B	F	B-C, C-D, D-F	30
4	D	A	D-C, C-A	20

Как видно из таблицы, одновременный выезд всех поездов со своих станций отправления невозможен ввиду возможной коллизии.

Также, будем считать все поезда не фиксированными и без приоритета проезда.

3.3.2. Введение данных

После ввода данных из предыдущего пункта «3.3.1. Подготовка данных», в консоли, из-за метода $toString()$ у объектов, можно наблюдать следующие результаты их обработки.

Граф путей (связей с другими станциями) задается следующим массивом:

```
PathGraph  =  [[0, 1, 1, 0, 0, 1, 0],
               [1, 0, 1, 0, 1, 0, 0],
               [1, 1, 0, 1, 0, 0, 0],
               [0, 0, 1, 0, 1, 1, 0],
               [0, 1, 0, 1, 0, 0, 1],
               [1, 0, 0, 1, 0, 0, 1],
               [0, 0, 0, 0, 1, 1, 0]]
```

Граф времени перемещения поездов между станциями задается следующим массивом:

```
TimeGraph =  [[0, 20, 10, 0, 0, 15, 0],
               [20, 0, 10, 0, 15, 0, 0],
               [10, 10, 0, 10, 0, 0, 0],
               [0, 0, 10, 0, 10, 10, 0],
               [0, 15, 0, 10, 0, 0, 15],
               [15, 0, 0, 10, 0, 0, 10],
               [0, 0, 0, 0, 15, 10, 0]]
```

Поезда задаются следующим списком:

```
ListOfTrain [numberOfTrains=4, listOfTrain=[
      Train [trainNo=1, stationBegin=0, stationEnd=4, tTravelMin=30],
      Train [trainNo=2, stationBegin=4, stationEnd=0, tTravelMin=30],
      Train [trainNo=3, stationBegin=1, stationEnd=5, tTravelMin=30],
      Train [trainNo=4, stationBegin=3, stationEnd=0, tTravelMin=20]]]
```

Поскольку поезда не зафиксированы и приоритет проезда отсутствует, то алгоритм произвольно устанавливает право проезда для каждого поезда. Таким образом, мы достигаем максимум разнообразия в популяции.

Затем формируется первоначальная популяция из 50 особей.

Критерий останова определяется в виде (14)

$$\sum_{i,j=0}^{m-1}(tEnd_j - tShift_j) - \sum_{i=0}^{n-1} 1{,}5 \cdot (tTravelMin_{\mathrm{i}}) \leq 0, \qquad (14)$$

где $n -$ количество поездов,

$m -$ количество генов в текущей особи для i поезда

3.3.3. Сравнение результатов

При сравнении результатов все 50 особей из начальной популяции не будут приведены, в виду того, что в этом нет смысла, но будет произведено сравнение одной из особей из начальной популяции с особью-решением. Сравнение выполнено в виде таблиц расписания 3, 4 для особи из начальной популяции и таблиц 5, 6 для особи-решения.

Предположим, что начальная точка во времени это 12:00 и все поезда отправились в путь в этот момент.

Генотип особи, которую сформировал алгоритм в начальной популяции:

TimeLine = [
(trNumber=3, tShift=0, tFinish=10, stationBegin=1, stationEnd=2, state=m),
(trNumber=1, tShift=0, tFinish=10, stationBegin=0, stationEnd=5, state=m),
(trNumber=3, tShift=10, tFinish=10, stationBegin=2, stationEnd=3 , state=m),
(trNumber=2, tShift=0, tFinish=15, stationBegin=4, stationEnd=1 , state=m),
(trNumber=2, tShift=15, tFinish=20, stationBegin=1, stationEnd=0 , state=m),
(trNumber=4, tShift=0, tFinish=10, stationBegin=3, stationEnd=5 , state=m),
(trNumber=1, tShift=15, tFinish=10, stationBegin=5, stationEnd=6, state=m),
(trNumber=4, tShift=10, tFinish=5, stationBegin=5, stationEnd=5 , state=w),
(trNumber=4, tShift=15, tFinish=5, stationBegin=5, stationEnd=0 , state=m),
(trNumber=3, tShift=20, tFinish=10, stationBegin=3, stationEnd=4, state=m),
(trNumber=1, tShift=25, tFinish=15, stationBegin=6, stationEnd=4, state=m),
(trNumber=3, tShift=30, tFinish=5, stationBegin=4, stationEnd=4, state=w),
(trNumber=3, tShift=35, tFinish=5, stationBegin=4, stationEnd=4, state=w),
(trNumber=3, tShift=45, tFinish=10, stationBegin=4, stationEnd=6, state=m),
(trNumber=3, tShift=55, tFinish=10, stationBegin=6, stationEnd=5, state=m)]

Генотип особи из решения после срабатывания критерия останова:

TimeLine = [
(trNumber=1, tShift=10, tFinish=10, stationBegin=2, stationEnd=1, state=m),
(trNumber=4, tShift=0, tFinish=10, stationBegin=3, stationEnd=2, state=m),
(trNumber=2, tShift=0, tFinish=15, stationBegin=4, stationEnd=1, state=m),
(trNumber=3, tShift=20, tFinish=10, stationBegin=3, stationEnd=5, state=m),
(trNumber=1, tShift=0, tFinish=10, stationBegin=0, stationEnd=2, state=m),
(trNumber=1, tShift=20, tFinish=15, stationBegin=1, stationEnd=4, state=m),
(trNumber=3, tShift=0, tFinish=10, stationBegin=1, stationEnd=2, state=m),
(trNumber=4, tShift=10, tFinish=10, stationBegin=2, stationEnd=0, state=m),
(trNumber=3, tShift=10, tFinish=10, stationBegin=2, stationEnd=3, state=m),
(trNumber=2, tShift=15, tFinish=20, stationBegin=1, stationEnd=0, state=m)]

Таблица 3 Расписание для поездов 1 и 2 из начальной популяции

Поезд 1			
Время отправления	Станция отправления	Станция пребывания	Затраченное время
12:00	A	F	15
12:15	F	G	10
12:25	G	E	15
tTravelMin	30		
Затраченное всего	40		
Крит.останова	$40 - 1{,}5 \cdot 30 \leq 0$		
Поезд 2			
Время отправления	Станция отправления	Станция пребывания	Затраченное время
12:00	E	B	15
12:15	B	A	20
tTravelMin	30		
Затраченное всего	35		
Крит.останова	$35 - 1{,}5 \cdot 30 \leq 0$		

Таблица 4 Расписание для поездов 3 и 4 из начальной популяции

Поезд 3			
Время отправления	Станция отправления	Станция пребывания	Затраченное время
12:00	B	C	10
12:10	C	D	10
12:20	D	E	10
--	E	E	10 (ожидание)
12:40	E	G	15
12:55	G	F	10
tTravelMin	30		
затраченное	65		
Крит.останова	$65 - 1{,}5 \cdot 30 > 0$		
Поезд 4			
Время отправления	Станция отправления	Станция пребывания	Затраченное время
12:00	D	F	10
--	F	F	5 (ожидание)
12:15	F	A	15
tTravelMin	20		
затраченное	30		
Крит.останова	$30 - 1{,}5 \cdot 20 \leq 0$		

Таблица 5 Расписание для поездов 1 и 2 из особи-решения

Поезд 1			
Время отправления	Станция отправления	Станция пребывания	Затраченное время
12:00	A	C	10
12:15	C	B	10
12:25	B	E	15
tTravelMin	30		
Затраченное всего	35		
Крит.останова	$35 - 1{,}5 \cdot 30 \ \leq 0$		
Поезд 2			
Время отправления	Станция отправления	Станция пребывания	Затраченное время
12:00	E	B	15
12:15	B	A	20
tTravelMin	30		
Затраченное всего	35		
Крит.останова	$35 - 1{,}5 \cdot 30 \ \leq 0$		

Таблица 6 Расписание для поездов 3 и 4 из особи-решения

Поезд 3			
Время отправления	Станция отправления	Станция пребывания	Затраченное время
12:00	B	C	10
12:10	C	D	10
12:20	D	F	10
tTravelMin	30		
затраченное	30		
Крит.останова	$30 - 1{,}5 \cdot 30 \leq 0$		
Поезд 4			
Время отправления	Станция отправления	Станция пребывания	Затраченное время
12:00	D	C	10
12:15	C	A	10
tTravelMin	20		
затраченное	20		
Крит.останова	$20 - 1{,}5 \cdot 20 \leq 0$		

3.3.4. Выводы

Можно заметить, что длина генотипа особи после работы генетического алгоритма гораздо короче, чем у особи, взятой из начальной популяции. Однако это еще не гарантирует того, что она лучше начальной. Вывод о качестве работы можно сделать о срабатывании критерии останова именно на особи-решении.

Из таблиц 3-6 видно, что с особью из начальной популяции, относительно особи из решения, повезло. Алгоритм сгенерировал хорошее решение относительно других особей из начальной популяции, так, что особь не прошла по критерию останова из-за плохой совокупности генов, описывающих третий поезд.

В результате работы алгоритма выяснилось, что общее время поездов в пути в особи из начальной популяции составляет 170 минут, а общее время в пути для особи-решения 120 минут. Возможно, для заданных параметров критерий останова является слишком мягким, и если бы его задать более строгим, то, возможно, получилось бы лучшее решение.

По результатам демонстрации алгоритма можно сделать вывод, что такие начальные параметры как размер транспортной сети, количество поездов, траектория их передвижения по этой сети и, главным образом, сам критерий останова (который тоже зависит от этих факторов) оказывают прямое влияние на полученное расписание движения поездов.

Заключение

По мере выполнения научно-исследовательской работы стало ясно, что выбранная тема очень обширная. Было прийто решение более подробно рассмотреть проблему генерации расписания движения поездов без проблемы подвязки локомотивов.

В ходе научно-исследовательской работы проведен анализ генетических алгоритмов и их применения в задачах условной оптимизации. Результатом научно-исследовательской работы является программный прототипа решения.

Несомненно, проблема расписания поездов - сложная и трудоемкая задача. Отсутствие специализированного программного обеспечения, которое бы учитывало все нюансы поставленной задачи таких как, например, пропускной способностью линии и другими эксплуатационными ограничениями, делает генетические алгоритмы одним из лучших подходов к эффективному решению такой задачи.

Предлагаемый в этой работе подход к получению расписания с помощью генетического алгоритма в виде программного прототипа решения может быть улучшен:

1. Возможно добавление графической оболочки, которая, в свою очередь, повысит дружелюбность интерфейса к пользователю и обеспечит быстрый ввод начальных данных;
2. Доработка механизма создания начальной популяции, которая будет включать в себя поезда с уже готовым расписанием, то есть те поезда, у которых заданы временя отправления и прибытия;
3. Доработка и уточнение критерия останова;
4. Доработка алгоритма: дать возможность делать умышленные остановки на станциях, что является критичным для пассажирских поездов;
5. Разработка и добавление логики касательно разрешения проблемы подвязки локомотивов;

6. Добавление более подробного логгирования позволит оптимизировать и легче дорабатывать алгоритм.

В дальнейшем, полученный программный прототип решения может быть также применен для составления расписания помимо реальной железнодорожной сети, но и любой другой транспортной сети, ограниченной какими-либо факторами.

Список литературы

1. A. Caprara, M. Fischetti, and P. Toth. Modeling and solving the train timetabling problem. Operations Research, 2002.
2. Goldberg D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley Publishing Company, 1989
3. Melanie Mitchell. An Introduction to Genetic Algorithms. — MIT Press, 1998
4. X. Cai, C. J. Goh. A fast heuristic for the train scheduling problem. Computers and Operations Research, 1994.
5. P. Tormos, A. Lova, F. Barber, L. Ingolotti, M. Abril, M.A. Salido. A Genetic Algorithm for Railway Scheduling Problems, Springer-Verlag Berlin Heidelberg, 2008.
6. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. 2006
7. Н.А.Кузнецов, Ф.Ф. Пащенко, Н.Г. Рябых, Е.М.Захарова, И.К.Минашина. Алгоритмы оптимизации в задачах планирования на рельсовом транспорте, Информационные процессы, Том 14, № 4, 2014, стр. 307–318, 201. [Electronic resource]. Mode of access: http://www.jip.ru/2014/307-318-2014.pdf
8. A. Caprara, M. Monaci, P. Toth, and P. Guida. A lagrangian heuristic algorithm for a real -world train timetabling problem. Discrete Applied Mathematics, 2006.
9. M. Carey and D. Lockwood. A model, algorithms and strategy for train pathing. The Journal of the Operational Research Society, 1995.
10. L. Kroon and L. Peeters. A variable time model for cycling railway timetabling. Transportation Science, 37(2):198–212, 2003.
11. Генетический алгоритм [Electronic resource]. Mode of access: http://www.machinelearning.ru/wiki/index.php?title=%D0%93%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC.

12. Г. К. Вороновский. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности, 1997
13. Генетические алгоритмы [Electronic resource]. Mode of access: http://qai.narod.ru/GA/index.html
14. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн Алгоритмы: построение и анализ, 2006
15. Емельянов В. В., Курейчик В. В., Курейчик В. М. Теория и практика эволюционного моделирования, 2003
16. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы: Учебное пособие, 2006.
17. C. Liebchen. Periodic Timetable Optimization in Public Transport. 2006.
18. D. Jovanovic and P. T. Harker. Tactical scheduling of rail operations, 1991
19. Люк С. Основы метаэвристик., 2009
20. Holland J. H. Adaptation in Natural and Artificial Systems: an introductory analysis with applications to biology, control, and artificial intelligence / University of Michigan Press. 1975
21. Dandy. G. C. An improved genetic algorithm for pipe network optimization, 1996
22. Verma A. Scaling genetic algorithms using mapreduce / Llora X., Goldberg D., Campbell R. 2009.
23. Играем с генетическими алгоритмами. [Electronic resource]. Mode of access: https://habr.com/post/246951/
24. Технологии анализа данных. [Electronic resource]. Mode of access: https://basegroup.ru/community/articles
25. Генетический алгоритм. Просто о сложном. [Electronic resource]. Mode of access: https://habr.com/post/128704/