|  |  |  |
| --- | --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  РОССИЙСКОЙ ФЕДЕРАЦИИ | | |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  «Пермский государственный национальный исследовательский университет» | | |
| УДК 004.023 | Кафедра математического обеспечения  вычислительных систем | |
| **Автоматическое составление учебных расписаний**  *Выпускная квалификационная работа* | | |
|  | | Работу выполнил студент группы ПМИ-1,2-2014 НБ 4 курса механико-математического факультета  Макурин Роман Анатольевич  «\_\_\_» \_\_\_\_\_\_\_\_2018 г. |
| Научный руководитель:  к.т.н., доцент кафедры МОВС  Городилов Алексей Юрьевич  «\_\_\_» \_\_\_\_\_\_\_\_2018 г. |
| Пермь 2018 | | |

**Аннотация**

Макурин Р.А. «Автоматическое составление учебных расписаний».

В работе описан подход к составлению учебного расписания в школе на основе генетических алгоритмов. Критически важные, универсальные требования к расписанию описываются на языке теории графов. При кодировании особей используется оригинальный подход на основе промежуточного представления, позволяющего применять стандартные генетические операторы. Промежуточное представление трансформируется в корректное расписание с помощью жадного алгоритма раскраски графа. Прочие, индивидуальные требования, описываются в виде гибко настраиваемой функции приспособленности, что делает возможной быструю адаптацию предложенного алгоритма к условиям конкретного учебного заведения. Алгоритм реализован в виде программы составления учебного расписания. В работе приведены результаты тестирования программы на реальных данных, показана корректность и преимущества предложенного подхода, отмечены пути развития.

Работа состоит из введения, обзора аналогичных систем, постановки задачи, описания и реализации генетических алгоритмов, заключения и приложений.

Всего страниц – 56.

Иллюстраций – 19.

Таблиц – 5.

Приложений – 2.

Содержание

[Введение 5](#_Toc517693183)

[Глава 1. Задача составления расписания 8](#_Toc517693184)

[1.1 Обзор основных подходов к решению задачи составления учебного расписания. 8](#_Toc517693185)

[1.1.1 Метод имитации отжига 9](#_Toc517693186)

[1.1.2 Мультиагентный (многоагентный) метод 9](#_Toc517693187)

[1.1.3 Метод генетического алгоритма 10](#_Toc517693188)

[1.2 Обзор и анализ аналогов 10](#_Toc517693189)

[1.2.1 ХроноГраф 3.0 Мастер 10](#_Toc517693190)

[1.2.2 aSc Расписание 2015 12](#_Toc517693191)

[1.2.3 1С:Автоматизированное составление расписания. Школа 14](#_Toc517693192)

[1.2.4 АВТОРасписание 16](#_Toc517693193)

[1.2.5 Выводы 17](#_Toc517693194)

[1.3 Описание предметной области 18](#_Toc517693195)

[1.4 Дополнительные условия задачи 19](#_Toc517693196)

[Глава 2. Теоретическое описание решения 20](#_Toc517693197)

[2.1 Формальная постановка задачи 20](#_Toc517693198)

[2.2 Описание генетического алгоритма 21](#_Toc517693199)

[2.3 Модели генетических алгоритмов 22](#_Toc517693200)

[2.4 Применение генетического алгоритма к задаче составления расписания 24](#_Toc517693201)

[2.5 Принципы селекции в генетических алгоритмах 30](#_Toc517693202)

[2.6 Варианты выполнения оператора кроссовера 31](#_Toc517693203)

[2.7 Оператор мутации 32](#_Toc517693204)

[2.8 Стратегии формирования нового поколения 32](#_Toc517693205)

[2.9 Оптимальность расписания 33](#_Toc517693206)

[Глава 3. Практическое решение задачи составления расписаний 34](#_Toc517693207)

[3.1 Функция оценки (фитнесс-функция) 34](#_Toc517693208)

[3.2 Формирование начальной популяции 35](#_Toc517693209)

[3.3 Инструменты реализации 36](#_Toc517693210)

[3.4 Структура программы 36](#_Toc517693211)

[3.5 Интерфейс пользователя 38](#_Toc517693212)

[3.6 Статистика работы генетического алгоритма 40](#_Toc517693213)

[3.7 Результат работы генетического алгоритма 42](#_Toc517693214)

[Заключение 43](#_Toc517693215)

[Список литературы 45](#_Toc517693216)

[Приложение А. Сгенерированное расписание 47](#_Toc517693217)

[Приложение Б. Листинг основных классов программы 50](#_Toc517693218)

# Введение

Составление расписания – одна из наиболее актуальных задач в планировании и оптимизации процесса обучения в учебных заведениях. От того, насколько хорошо составлено расписание, зависит эффективность работы преподавателей, усвоение учебного материала учениками, рациональное использование материальных ресурсов.

Составлением школьного расписания занимаются люди, ответственные за это – в школе это диспетчер по расписанию. Он, имея под руками учебный план, учебную нагрузку и несколько других регламентирующих документов, составляет учебное расписание, следуя которому, работают преподаватели и обучаются ученики. Качественно составленное расписание обеспечит высокую эффективность обучения.

Проблема составления расписания занятий является предметом исследования в работах многих авторов. В современных условиях возникает необходимость использования автоматизированных средств планирования и составления расписания в учебных заведениях.

Почему же в век высоких технологий, процесс составления школьного расписания до сих пор остаётся ручным трудом? Потому что точных алгоритмов, решающих данную задачу за приемлемое время не существует. Кроме того, подобная задача не обладает чёткой формулировкой критериев оптимальности.

Процесс составления расписания основан на анализе большого количества информации и требует значительных трудозатрат. Эффективное управление ресурсами является одной из самых сложных и самых важных задач, стоящих перед диспетчерами.

Актуальность автоматизации составления расписаний определяется ростом требований к качеству обучения, планированию работы учеников, рациональному использованию аудиторного фонда, а также учетом дополнительных параметров оптимизации. Задача составления учебного расписания относится к общей теории расписаний. Ее можно классифицировать по типу целевой функции как задачу многокритериальной оптимизации, критерии которой могут меняться с течением времени.

*Объектом* исследования данной работы является учебное расписание.

На практике для решения этой задачи применяют различные приближенные, эвристические алгоритмы, так что программы для составления расписания на настоящий момент все-таки существуют. Некоторые из них довольно неплохо решают поставленную задачу, даже генерируя приемлемые решения, однако, они не учитывают все составляющие школьного образовательного процесса, так что диспетчеру по расписанию приходится затем в «ручном» режиме (в некоторых случаях даже на бумаге) корректировать составленное программным средством расписание.

При автоматизации составления учебного расписания возникают такие проблемы, как многокритериальность и эвристический характер многих критериев.

*Предметом* исследования данной работы является автоматическое составление учебных расписаний.

В данной работе выбран подход на основе генетического алгоритма, (который как раз относится к классу эвристических), так как генетический алгоритм успешно применяется в областях, где критерий оптимальности задан нечётко.

В связи с этим *целью* этой работы является разработка программы автоматического составления учебного расписания с применением генетического алгоритма.

Для достижения сформулированной цели необходимо решить следующие *задачи*:

1. обзор литературы по генетическим алгоритмам;
2. обзор программ аналогов по составлению расписания;
3. определение структуры для хранения расписания;
4. разработка генетического алгоритма составления расписания;
5. разработка программы;
6. анализ результатов работы генетического алгоритма.

В ходе решения поставленных задач предполагается использовать следующие методы исследования: сравнительный анализ существующих решений, составление требований к разрабатываемой системе, объектно-ориентированное программирование.

# Задача составления расписания

В данной главе рассмотрены следующие этапы исследования предметной области:

1. обзор основных подходов к решению задачи составления учебного расписания;
2. обзор аналогичных решений;
3. формальное описание предметной области.

## Обзор основных подходов к решению задачи составления учебного расписания.

В связи с тем, что это – задача по типу многокритериальной оптимизации оценочной функции, можно выделить несколько подходов к ее решению:

1. классические методы оптимизации;
2. эвристические методы оптимизации;
3. методы перебора.

Для решения задачи составления расписания классические методы оптимизации трудно применимы в силу следующих особенностей: большая размерность, эвристический характер многих критериев и широкая область поиска решения. При использовании метода полного перебора сложность задачи составления расписания будет зависеть от количества всех возможных решений задачи, конкретно от числа уроков. Задача составления расписания является NP-полной, поэтому полный перебор потребует экспоненциальное время работы. Соответственно это приведет к тому, что результат может быть получен через несколько лет или даже столетий. Поэтому далее будут рассмотрены эвристические методы решения задачи составления учебного расписания. Несмотря на то, что эвристические методы не обеспечивают стопроцентную вероятность нахождения оптимального решения, локальные экстремумы могут считаться приемлемыми в данной задаче.

### Метод имитации отжига

Алгоритм основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Предполагается, что атомы уже выстроились в кристаллическую решётку, но ещё допустимы переходы отдельных атомов из одной ячейки в другую. Процесс протекает при постепенно понижающейся температуре. Переход атома из одной ячейки в другую происходит с некоторой вероятностью, причём вероятность уменьшается с понижением температуры. Устойчивая кристаллическая решётка соответствует минимуму энергии атомов, поэтому атом либо переходит в состояние с меньшим уровнем энергии, либо остаётся на месте. Алгоритм имитации отжига похож на градиентный спуск, но за счёт случайности выбора промежуточной точки должен будет попадать в локальные минимумы реже, чем градиентный спуск.

### Мультиагентный (многоагентный) метод

Представляет собой множество взаимодействующих интеллектуальных единиц (агентов), каждый из которых выполняет определённые функции, направленные на достижение своих целей, а совместное поведение таких агентов должно обеспечить достижение оптимального состояния для всей системы.

Основными характеристиками агентов являются:

1. автономность (независимость);
2. ограниченность представления – ни у одного из агентов нет представления обо всей системе, или система слишком сложна, чтобы знание о ней имело практическое применение для агента;
3. децентрализация – нет агентов, управляющих всей системой.

Главное достоинство многоагентных систем — это гибкость. Многоагентная система может быть дополнена и модифицирована без переписывания значительной части программы. При составлении расписания учителей и учеников можно заменить на агентов, которые взаимодействуют между собой, пытаясь найти приемлемое время и место занятия.

### Метод генетического алгоритма

Генетический алгоритм (ГА) – это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе [2]. Метод характеризуется следующими операциями: наследование, мутация, отбор и кроссовер (скрещивание). Отличительной чертой генетического алгоритма является акцент на использование оператора кроссовера, который производит операцию рекомбинации особей, роль которой аналогична роли скрещивания в живой природе.

Особь характеризуется набором некоторых признаков – генов. Формируется начальное поколение особей. Наиболее приспособленные особи имеют больше шансов на выживание. Затем в результате отсеивания оставшиеся особи скрещиваются и производят потомство, адаптированное к текущим условиям. Потомство представляет собой различную комбинацию генов родителей, все зависит от оператора кроссовера. У некоторых особей, с небольшой вероятностью, возможны мутации – это случайное изменение одного или нескольких генов. Затем особи-потомки (а в некоторых случаях и родители) представляют собой новое поколение, и процесс повторяется сначала, пока не будет достигнуто желаемое количество поколений или несколько поколений подряд не будут иметь существенных изменений в генах.

## Обзор и анализ аналогов

### ХроноГраф 3.0 Мастер

ХроноГраф 3.0 Мастер является бесплатной программой для составления школьного расписания. Интерфейс программы предоставлен на рисунке Рисунок 1.

Пакет адресован руководителям средних общеобразовательных учреждений, отвечающим за организацию и контроль качества учебно-воспитательного процесса. Включает в себя [2]:

1. Учет специфики конкретного общеобразовательного учреждения и подготовку следующих исходных данных:
2. общие сведения и структура учебного заведения в соответствии со стандартами;
3. список учеников;
4. недельная сетка часов;
5. данные о преподавателях;
6. аудиторный фонд;
7. распределение учебной нагрузки.

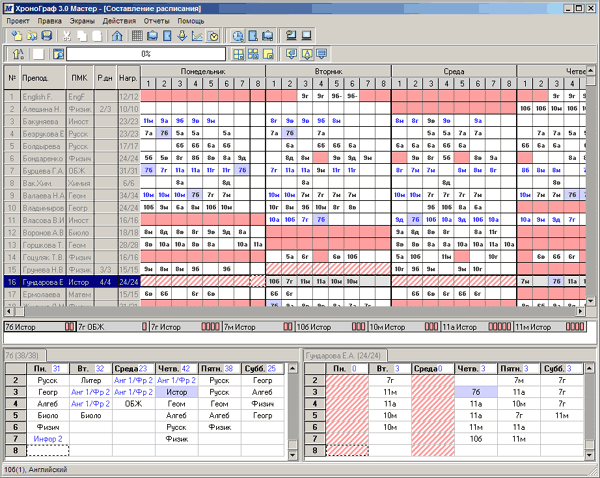


Рисунок 1 – Хронограф 3.0 Мастер

1. Создание методически выдержанного школьного расписания уроков на основе:
2. формирования и оптимизации графиков работы преподавателей с учетом нагрузки;
3. работы со всеми срезами расписания;
4. визуального контроля за графиками загруженности учащихся по неделе;
5. динамического контроля за распределением аудиторного фонда.
6. Оперативное ведение расписания занятий в течение всего учебного периода с возможностями:
7. реализации ежедневных замен преподавателей;
8. привлечения к проведению замен сотрудников школы, непосредственно не ведущих учебные занятия;
9. сохранения информации и ведение базы данных о произведенных заменах в течение всего учебного периода.
10. Подготовку и возможность настройки параметров печати выводных форм.

Данная программа имеет следующие преимущества:

1. просмотр нагрузки учителей;
2. возможность замены учителя в случае отсутствия;
3. высокая скорость работы.

Однако у этой программы есть и ряд недостатков, среди которых:

1. перегруженный интерфейс;
2. сложное управление;
3. расписание может быть составлено не оптимально;
4. неудобно вводить исходную информацию;
5. нет возможности разделить класс на группы.

### aSc Расписание 2015

Платная программа для составления расписания, интерфейс которой изображен на рисунке Рисунок 2. Облегчает составление расписания и помогает гарантировать успешную работу образовательного учреждения на протяжении всего учебного года.

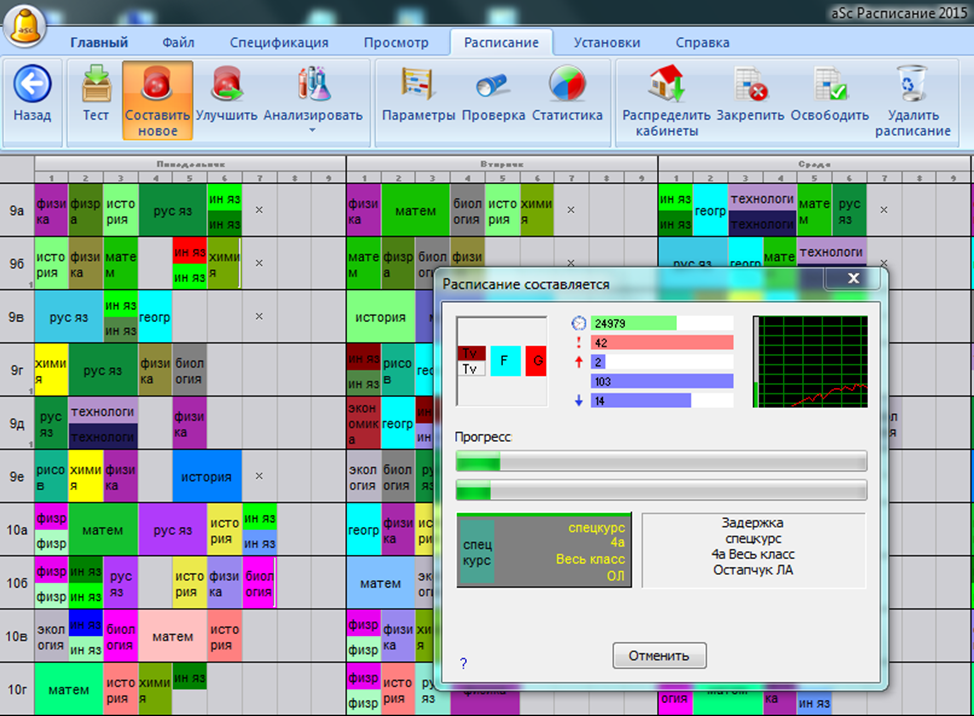


Рисунок 2 – aSc Расписание 2015

В данной программе можно выделить следующие положительные стороны:

1. высокая скорость работы;
2. интуитивно понятный интерфейс;
3. начальная страница, которая позволяет посмотреть примеры и обучает функционалу программы;
4. учет нагрузки преподавателей, учеников;
5. замена преподавателя в случае отсутствия;
6. возможность составлять курсы для нескольких учеников по выбору;
7. множество дополнительных функций, которые, однако, нужны лишь в тех случаях, когда происходит ручное составление расписания;
8. мобильное приложение;
9. по словам разработчиков «Умный алгоритм», который сам подстраивается под расписание.

Но расписание все так же получается не оптимальным в виду присутствия окон, то есть незанятый урок в расписании учеников.

### 1С:Автоматизированное составление расписания. Школа

Аналогично предыдущему является платным решением, изображена на рисунке Рисунок 3. Программа «1С:Автоматизированное составление расписания. Школа» предназначена для составления «умного» расписания, индивидуальных траекторий и дополнительной занятости учащихся с учетом основного расписания и аудиторного фонда в школах. Гибкие настройки программы позволяют успешно использовать ее в учреждениях со сложной структурой: образовательные комплексы, объединяющие детские сады, школы; центры творчества и дополнительного образования детей; частные школы или центры развития с индивидуальным графиком для каждого ребенка [3].

В программе учтены требования СанПиН, ФГОС к составляемым расписаниям (состав, порядок следования уроков, оптимальная нагрузка в течение дня, максимальная нагрузка и т.д.).

Программа позволяет:

1. составлять расписание различной сложности в автоматическом, ручном и смешанном режимах;
2. вести несколько сеток звонков;
3. составлять расписание без привязки к сетке звонков, назначая для каждого занятия продолжительность и определяя произвольное время начала;
4. выводить расписание на печать по классам, преподавателям и помещениям, с возможностью быстрых настроек выводимой информации;
5. копировать расписание прошлых периодов и корректировать его;

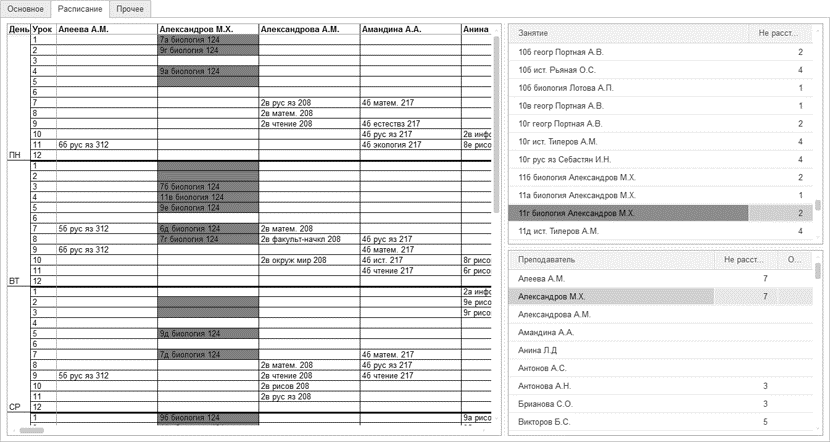
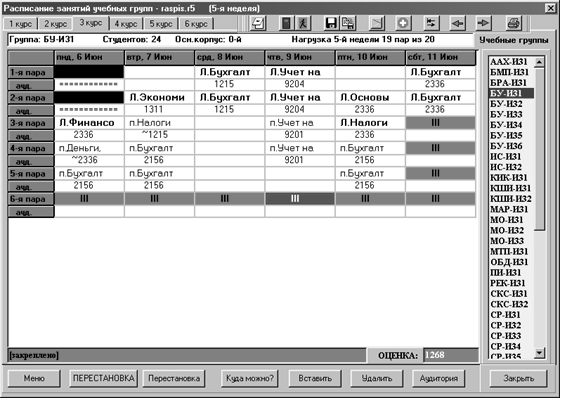


Рисунок 3 – 1С:Автоматизированное составление расписания. Школа

1. учитывать требования СанПиН и ФГОС по последовательности проведения занятий, максимальной дневной нагрузке с учетом сложности занятий;
2. вводить и учитывать сложность предметов/занятий/дисциплин в баллах;
3. учитывать пожелания и возможностей учителей, классов учащихся, помещений;
4. учитывать разбиение на подгруппы;
5. составлять индивидуальные траектории для групп и отдельных учащихся;
6. строить расписание для 1, 2-х и более смен;
7. автоматически проверять расписание на ошибки, удобно их устранять;
8. корректировать расписание с нужной периодичностью, сравнивать расписания;
9. импортировать и экспортировать данные из «1С:Общеобразовательное учреждение»;
10. формировать учебный план на основе готового шаблона базисного учебного плана, рекомендованного Министерством образования и науки РФ;
11. составлять несколько расписаний и выбирать лучшее;
12. подбирать и вести замены;
13. вести учет аудиторного фонда образовательной организации;
14. формировать отчеты об использовании помещений и проведенных занятиях.

### АВТОРасписание

Система "АВТОРасписание", представленная на рисунке Рисунок 4, предназначена для быстрого, удобного и качественного составления расписаний занятий и сопровождения их в течение всего учебного года [5].

 Рисунок 4 – АВТОРасписание

Программа позволяет:

1. работать с расписанием любой степени сложности;
2. составлять недельные, двухнедельные, семестровые расписания, а также расписания сессий заочников с плавающим графиком обучения и расписания разовых мероприятий;
3. помогает легко строить, корректировать и распечатывать расписание в виде удобных и наглядных документов.

### Выводы

В результате анализа была составлена сравнительная таблица Таблица 1.

Современные системы составления расписания не предоставляют информации об использующемся в них алгоритме, но позволяют:

1. задавать исходные данные (список классов, учителей, учебную нагрузку);
2. генерировать расписание;
3. редактировать расписание.

Таблица - Сравнение аналогов

|  | Хронограф 3.0 Мастер | aSc Расписание 2015 | 1С:Автоматизированное составление расписания. Школа | АВТОРасписание |
| --- | --- | --- | --- | --- |
| Учет учебного плана | + | + | + | + |
| Учет пересечений расписания | +/- | + | + | + |
| Учет индивидуальных расписаний | - | - | + | - |
| Возможность ручной корректировки расписания | + | + | + | + |
| Динамически настраиваемые правила генерации | - | - | - | - |
| Удобный интерфейс | - | - | - | - |

К главным недостаткам программ можно отнести отсутствие динамически настраиваемых правил генерации, неоптимальность получаемого расписания, неудобная структура просмотра расписания, интерфейс программ выглядит перегруженным.

При анализе методов, с помощью которых можно решить задачу автоматического составления расписания, было принято решение использовать генетический алгоритм. Генетические алгоритмы более развиты, а также известны примеры их успешного применения для решения задач с нечетким критерием оптимальности. ГА по сравнению с методом отжига способен скрещивать особи между собой и выявлять наиболее пригодные для дальнейшей популяции, в то время как в другом алгоритме имеется только одна особь. В данной задаче скрещивание особей, в данном случае разных видов расписаний, позволяет получить новое расписание, способное удовлетворять сразу нескольким критериям. Также необходимо учитывать вычислительную сложность задачи, а генетический алгоритм имеет несколько особей и вычисление фитнесс-функции может выполняться параллельно для каждой особи. Метод отжига не обладает такой возможностью в виду работы только с одной особью. Многоагентный метод показал непригодность для решения задачи составления расписания [5].

## Описание предметной области

Выделим основные понятия учебного процесса:

1. *расписание* – график, таблица, содержащая указания о времени, месте и последовательности школьных занятий;
2. *учебный* *план* – это документ учебного заведения, определяющий полный перечень предметов, изучаемых в учебном заведении; количество часов по каждому предмету за все время обучения и на изучение предмета в каждом классе; количество часов в неделю на изучение каждого предмета;
3. *позиция учебного плана* – сочетание информации о преподавателе, классе, предмете и количестве часов, отведённом на его изучение в неделю;
4. *диспетчер по расписанию* – лицо, отвечающее за организацию учебно-воспитательного процесса, формирующее учебное расписание на основе учебного плана;
5. *позиция расписания* – минимальная единица расписания, включающая в себя фиксированные аудиторию, преподавателя, предмет и класс.

## Дополнительные условия задачи

В результате изучения информации о предметной области можно выделить следующие требования к школьному расписанию, установленные СанПиНом [6]:

1. не допускать превышения количества часов, указанных в  
   таблице таблица 2;
2. чередовать естественно-математические и гуманитарные предметы;
3. увеличивать учебную нагрузку к середине недели, а затем уменьшать её.

Таблица 2 – Максимально допустимая нагрузка

|  |  |  |
| --- | --- | --- |
| Классы | Максимальная недельная нагрузка в часах | |
| При 6-ти дневной неделе | При 5-дневной неделе |
| **3-летняя начальная школа:** | | |
| 1 - 3 | 25 | 22 |
| **4-летняя начальная школа:** | | |
| 1 | 22 | 20 |
| 2 - 4 | 25 | 22 |
| 5 | 31 | 28 |
| 6 | 32 | 29 |
| 7 | 34 | 31 |
| 8 - 9 | 35 | 32 |
| 10 - 11 | 36 | 33 |

# Теоретическое описание решения

В данной главе рассматривается применение генетического алгоритма к задаче составления расписания, включающее следующие элементы:

1. формальное определение постановки задачи;
2. работа алгоритма жадной раскраски в виде блок-схемы и его описание;
3. проблема применения оператора скрещивания и её решение;
4. используемые операторы генетического алгоритма.

## Формальная постановка задачи

Расписание – это функция

(1)

где

– класс (группа)

– позиция расписания (время проведения)

– преподаватель

– предмет

– кабинет

Обязательные требования (расписание f – корректно, если):

1. из определения функции вытекает, что

существует не более одного значения

, (2)

то есть отсутствуют пересечения у классов;

1. существует не более одного значения :

, (3)

отсутствуют пересечения у преподавателей.

Необязательные требования:

1. Отсутствие окон, незанятых уроков у учеников и преподавателей;
2. Индивидуальные предпочтения учителей.

На вход программе поступает информация из учебного плана, а именно множество позиций учебного плана, а в результате работы получается расписание. Результатом формирования расписания школьных занятий являются различные таблицы общепринятой формы и содержания. Эти таблицы, собственно и являются школьным расписанием. Они содержат для каждого класса (группы) указание о последовательности занятий и месте их проведения. В каждой ячейке таблицы содержится тройка формата .

*Корректным* расписанием является расписание, удовлетворяющее всем обязательным требованиям.

*Оптимальным* расписанием считается такое расписание, которое удовлетворяет всем обязательным и нескольким необязательным требованиям. Чем большему числу необязательных требованиям удовлетворяет расписание, тем более оптимальным оно считается.

## Описание генетического алгоритма

Генетические алгоритмы обрели свою популярность благодаря работе Джона Холланда «Adaptation in Natural and Artificial Systems» (1975) [7], где он впервые вводит понятие «генетический алгоритм». До этого времени подобные алгоритмы назывались «симуляция эволюции», «стратегия эволюции», «искусственная эволюция». Описанный там алгоритм теперь называют классическим ГА.

Как уже было упомянуто выше генетические алгоритмы работают подобно эволюции в природе. Они используют такие понятия, как особь, которая является неким решением задачи, и функция приспособленности (fitness-function), которая определяет приспособленность особей-решений. Таким образом, лучшие особи имеют большую вероятность на скрещивание, а их потомство должно взять от родителей самое лучшее, то есть каждое новое поколение в среднем получается лучше предыдущего.

Генетический алгоритм – это итерационный процесс, который продолжается до тех пор, пока не выполнится критерий останова (например, заданное число поколений) [8].

На каждом поколении генетического алгоритма реализуется селекция (отбор), кроссовер и мутация. Данный алгоритм можно представить схемой, изображённой на рисунке Рисунок 5.

Каждая особь кодируется неким образом, это может быть битовая, численная или же символьная строка, где каждый символ является неким геном, а вместе они образуют генотип особи. В классическом ГА начальное поколение создаётся случайным образом. Численность поколения – это вариативная переменная, обычно фиксировано заданная в начале алгоритма.

Рисунок 5 – Схема работы классического генетического алгоритма

Начальная популяция

Селекция

Кроссовер

Мутация

Переход к новому поколению

Результат

Далее формируется промежуточная популяция – это особи, которые получили право размножаться и скрещиваться. Наиболее приспособленные особи имеют большую вероятность для скрещивания, они также получают возможность скрещиваться несколько раз.

После скрещивания происходят мутации, как уже упоминалось выше, случайные изменения одного или нескольких генов. Полученное поколение принимается за новое и алгоритм повторяется вновь.

## Модели генетических алгоритмов

Различные модели ГА рассматривались в работе [9].

**Классический ГА** был описан Джоном Холландом [7]. Согласно ему, популяция состоит из фиксированного числа хромосом и количества генов. На первоначальном этапе выполняется пропорциональный отбор, после чего случайным образом выбираются два родителя, которые скрещиваются одноточечным кроссовером, а к потомкам применяется одноточечная мутация. Потомки заменяют собой родителей в популяции. Вновь выполняется пропорциональный отбор, и алгоритм выполняется сначала, пока не выполнится критерий останова.

**Genitor**

Алгоритм был предложен Даррелом Уитли (Darrel Whitley) [10]. Основными отличиями от классического генетического алгоритма являются: пара родителей порождает лишь одного потомка, который, в свою очередь, заменяет худшую (а в некоторых вариациях одну из худших) особь; отбор особи для замены происходит по её рангу, а не по значению функции приспособленности.

**CHC**

СHC расшифровывается как Cross-population selection, Heterogenous recombination, Cataclysmic mutation. Был создан Эшелманом (L. Eshelman). Отличительными характеристиками являются:

1. при формировании следующего поколения выбирается фиксированное число особей среди родителей и детей;
2. для скрещивания выбирается случайная пара, однако, к скрещиванию допускаются пары, для которых Хеммингово расстояние велико;
3. при скрещивании используется модификация однородного кроссовера (HUX), при которой потомок наследует по половине генов от каждого родителя.

Размер популяции – около 50 особей, что приводит алгоритм в состояние, когда все особи слабо отличаются друг от друга. В случае возникновения таких ситуаций все особи, за исключением самой приспособленной, подвергаются сильной мутации (изменяется около 30% генов). Благодаря чему происходит перезапуск алгоритма, и он продолжает работу.

**Гибридный алгоритм (L. "Dave" Davis)**

Идея алгоритма заключается в сочетании ГА с некоторым классическим методом оптимизации. Каждый полученный потомок оптимизируется выбранным методом и включается в новую популяцию, что позволяет каждой особи достичь локального минимума, рядом с которым она находится. После этого к популяции применяется генетический алгоритм.

На практике гибридные алгоритмы оказываются очень удачными. В первую очередь это связано с большой вероятностью того, что одна из особей попадёт в область глобального максимума и после оптимизации окажется решением задачи.

Генетический алгоритм способен быстро найти во всей области поиска хорошие решения, но он может испытывать трудности в выделении из них наилучших вариантов. Сочетание двух алгоритмов способно использовать преимущества обоих.

**Островная модель**

Основное отличие отражено в названии. В начале алгоритма популяция делится на несколько «островов», на каждом из которых генетический алгоритм выполняется независимо от других. Раз в несколько поколений происходит незначительный обмен хорошими особями – миграция.

Островная модель позволяет запустить несколько копий генетический алгоритма параллельно. При разных запусках алгоритма получаются разные решения, которые в дальнейшем комбинируются друг с другом, что позволяет достичь лучших результатов.

## Применение генетического алгоритма к задаче составления расписания

Введем некоторые ограничения в решаемую задачу. Будем составлять школьное расписание и условимся, что каждый день у каждого класса не более шести уроков, а рабочая неделя состоит из пяти дней. Таким образом, количество уроков в расписании одного класса на неделю не превышает 30.

В генетическом алгоритме особь представляется некой последовательностью, называемой хромосомой, и описывает одно из возможных решений задачи. В решаемой задаче решением является некоторое расписание, то есть таблица, содержащая для каждого класса (группы) указание о последовательности занятий и месте их проведения.

Непосредственное кодирование расписания в виде хромосомы, при котором урок в расписании представляет собой ген, влечет появление большого количества «нежизнеспособных» особей в результате скрещивания и мутаций, которые будут описаны далее, в силу появления накладок в расписании преподавателей (когда предметы одного учителя попадают на одно и то же время), то есть не удовлетворяет второму обязательному условию корректности расписания (3). Поэтому в предлагаемом решении используется промежуточное представление.

Представим учебный план в виде графа. Граф , где -множество вершин, каждая вершина , – конкретный класс из множества классов , – конкретный преподаватель, из множества преподавателей , а – конкретный предмет, из множества предметов *S*. Вершины , являются смежными тогда, когда , или .

Для примера: Мария Ивановна ведет в 6А классе 4 часа русского и 5 часов литературы, следовательно, мы добавим в наш граф 9 вершин, которые представляют такое сочетание. Такую операцию проделаем для каждого сочетания, затем установим связи между вершинами. Смежными вершинами будут являться вершины, в которых совпадает преподаватель и/или класс.

Таким образом представим наш учебный план в виде сильно связного графа. Правильно раскрасим граф G (любые две смежные вершины будут окрашены в разные цвета). Тогда все вершины, покрашенные в один цвет, не вызывают конфликтов вида (2) и (3), и соответствующие им уроки можно провести одновременно. Назначая каждому цвету своё время (номер урока), получим корректное расписание.

Для правильной раскраски графа применим жадный алгоритм, блок-схема изображена на рисунке Рисунок 6, суть которого заключается в последовательном просмотре всех вершин и раскраске каждой вершины в первый возможный цвет. Естественно, что получаемая раскраска существенно зависит от последовательности просмотра вершин. Хромосома в ГА как раз и будет кодировать такую последовательность. Таким образом, любая особь в ГА кодирует корректное расписание (отсутствие ситуаций, когда один преподаватель по расписанию должен вести занятие сразу у двух классов или, наоборот, когда один класс должен находиться одновременно на двух занятиях), а далее получаемые расписания оцениваются с помощью прочих гибко настраиваемых критериев оптимальности.

Рисунок 6 – Блок-схема работы алгоритма раскраски

Цикл по всем вершинам

*C* := 1

Красим в цвет *C*

Смежные c вершины окрашены в другие цвета?

нет

*C* := *C*+1

Красим в цвет *C*

да

Конец

Начало

В нашей модели мы предположили по 5 учебных дней в неделю, в которых каждый день возможно до 6 учебных часов. Для того чтобы «уложить» раскрашенный граф в расписание необходимо чтобы число цветов не превышало 30.

Каждый цвет представляет собой определенное время проведения занятия, то есть единовременно могут идти только занятия, находящиеся в вершинах одного цвета. Создаем схему укладки раскрашенных вершин, для упрощения можно сказать что первый урок в понедельник — это вершины, окрашенные в первый цвет, следующий соответственно 2, третий урок в среду будет иметь уже вершину, окрашенную в цвет под номером 15, ну и 6 урок в пятницу имеет вершину, раскрашенную в цвет под номером 30. Эта привязка лишь условная, мы без проблем можем поменять цвета местами.

Что же касается генетического алгоритма, он будет помогать нам находить такой порядок раскраски, который даст наиболее оптимальное расписание. Рассмотрим пример, в котором имеется 3 класса, 3 преподавателя и 3 предмета. Особь представляет из себя последовательность раскраски графа, который изображен на рисунке Рисунок 7. Для примера возьмем тривиальный порядок раскраски от 1 до 6.

6А

Петрова

Русский

6В

Петрова

Русский

6А

Климова

Математика

6Б

Климова

Математика

6А

Иванов

Биология

6Б

Иванов

Биология

1

2

3

4

5

6

Рисунок 7 – Графовое представление учебного плана

В указанном порядке раскрасим вершины и в результате работы жадного алгоритма получим граф, изображенный на рисунке Рисунок 8.

6А

Петрова

Русский

6В

Петрова

Русский

6А

Климова

Математика

6Б

Климова

Математика

6А

Иванов

Биология

6Б

Иванов

Биология

Рисунок 8 - Раскрашенный граф

А в конечном итоге получаем расписание, показанное в таблице Таблица 3.

Таблица 3 - Полученное расписание

|  |  |  |  |
| --- | --- | --- | --- |
|  | 6А | 6Б | 6В |
| 1 | Математика  Климова | Биология  Иванов | Русский язык  Петрова |
| 2 | Русский язык  Петрова | Математика  Климова |  |
| 3 | Биология  Иванов |  |  |

Представление в виде прямого порядка раскраски графа интуитивно понятно, но приводит к проблемам применения операторов: в таком представлении особей чаще всего невозможно корректно применить классические операторы скрещивания и мутации. Рассмотрим это на примере:

Допустим родитель А имеет вид – 2 3 4 1. Родитель Б – 3 1 2 4. Применяя равномерный кроссовер получим:

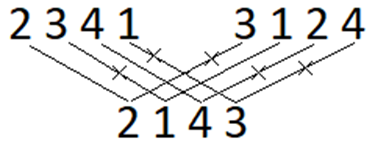


Рисунок 9 – Пример равномерного кроссовера

На рисунке рисунок 9 видно, что при скрещивании необходимо учитывать какие числа уже были в последовательности, и ко всему прочему последнее число не может быть получено ни от одного из родителей.

Аналогично при одноточечном кроссовере, получаются потомки: 2 3 2 4 и 3 1 4 1. Что никак не может являться порядком раскраски в виду наличия повторяющихся значений.

Для решения этой проблемы воспользуемся таким понятием, как фенотип. Фенотип – это набор значений, соответствующих данному генотипу, декодированная структура или множество параметров задачи. В нашем случае можно использовать следующий способ кодирования особей - каждый ген представляет из себя индекс вычеркивания из упорядоченного списка.

Например,

2 3 4 1

Выпишем упорядоченный список: 1 2 3 4.

Теперь проходим по всем числам особи и вычеркиваем соответствующие числа в упорядоченном списке и запоминаем индекс:

Таблица 4 - Пример построения фенотипа

|  |  |  |
| --- | --- | --- |
| Упорядоченный список | Исходное число | Результат |
| 1 2 3 4 | 2 | 2 |
| 1 3 4 | 3 | 2 2 |
| 1 4 | 4 | 2 2 2 |
| 1 | 1 | 2 2 2 1 |

Так мы преобразуем порядок 2 3 4 1 в комбинацию 2 2 2 1.

В результате такого представления при использовании оператора скрещивания все особи потомки будут получатся корректными. У генотипа накладывается следующее свойство – , где – значение i-ого разряда, *n* – общее число разрядов.

Имея учебный план, где описано число уроков по каждому предмету в неделю у каждого класса, будем создавать граф *G* и генерировать случайным образом фенотип, преобразуемый затем в генотип, который будет являться порядком раскраски вершин для жадного алгоритма.

## Принципы селекции в генетических алгоритмах

Селекция – это отбор тех особей, которые будут участвовать в создании следующего поколения. Существуют различные варианты отбора, рассмотрим основные из них [11].

**Принцип рулетки** – подобен колесу рулетки, где каждой особи сопоставляется сектор, пропорциональный её приспособленности, поэтому чем больше значение функции приспособленности, тем больше размер сектора рулетки, и тем больше вероятность быть выбранным. Однако у этого метода существует недостаток – особи с малым значением функции приспособленности очень быстро выходят из популяции, в результате чего генетический алгоритм сходится к локальному экстремуму. Для того, чтобы исключить преждевременную сходимость, используют альтернативные методы.

**Принцип турнирной селекции** – в этом случае все особи разбиваются на несколько групп, чаще всего на группы размером 2-3 особи, затем в группе определяется лидер, который и проходит для дальнейшего создания промежуточной популяции. Исследования подтверждают, что турнирный метод действует эффективнее, чем метод рулетки [12].

**Принцип ранговой селекции** – особи сортируются по значению функции приспособленности, выстраиваются по рангам. Затем задаётся некая функция, которая рассчитывает количество копий особи пропорционально значению её приспособленности.

**Принцип элитарной стратегии** – в случае с классическим генетическим алгоритмом существует вероятность исключения из поколения самой приспособленной особи. Чтобы предотвратить это, используют элитарную стратегию, при которой самая приспособленная особь включается в следующую популяцию.

**Принцип частичной замены популяции** – выбираются две особи и скрещиваются между собой, далее особь-потомок с наибольшим значением функции приспособленности заменяет собой родителя с наименьшим значением приспособленности. В результате получается новое поколение с одной изменённой особью. В различных вариациях такого алгоритма встречается разное число скрещиваний, но обычно это замена одной особи родителя, вследствие одного скрещивания. Поэтому данный метод еще называют генетическим алгоритмом с зафиксированным состоянием.

В реализованном генетическом алгоритме для отбора особей был выбран рулеточный отбор, при котором более приспособленная особь имеет большую вероятность на скрещивание.

## Варианты выполнения оператора кроссовера

Немаловажную роль играет и оператор кроссовера (он же оператор скрещивания). Чаще всего на практике встречается одноточечный кроссовер, при котором выбирается случайная точка разрыва генов родителей и в результате получается две дочерние особи, одна из которых содержит начало одного родителя и конец другого, а вторая наоборот, конец первого и начало второго.

Аналогично одноточечному кроссоверу, выполняется двухточечный, только в этом случае особь представляет собой закольцованную последовательность, в которой случайным образом выбирается два разрыва, и из полученных последовательностей образуется две дочерние особи по аналогии с одноточечным.

Также встречается еще и однородный кроссовер, при котором каждый ген родительской особи наследуется с некой вероятностью (обычно 0.5), иначе наследуется ген другой родительской особи. Оставшиеся родительские гены наследуются второй дочерней особью.

Для скрещивания был выбран одноточечный кроссовер, так как в решаемой задаче именно он позволяет потомку частично наследовать от родителей порядок раскраски, так как нам важно перенимать порядок раскраски родителей. При использовании равномерного кроссовера такого не происходит.

## Оператор мутации

Согласно схеме классического ГА с некоторой малой заданной вероятностью выполняется оператор мутации. Оператор мутации инвертирует значение одного гена на противоположное (в некоторых случаях несколько генов).

Одноточечный оператор мутации представляет собой выбор гена и обмен его на расположенный рядом ген. Двухточечный оператор мутации выбирает два случайных гена и проводит обмен между генами и их соседями. Развитием двухточечного оператора мутации является многоточечный (или n-точечный) оператор мутации. В этом случае происходит последовательный обмен генов, расположенных правее точек разреза друг с другом в порядке их расположения.

В разработанной программе для оператора мутации опытным путём установлено следующее правило: с вероятностью 5% происходит обмен четырех генов потомка между собой.

## Стратегии формирования нового поколения

Выделяют два типа формирования нового поколения:

1. Дочерние особи заменяют родителей;
2. Новое поколение формируется из всех полученных особей путём селекции.

Использование второго типа позволяет не терять особи с наибольшим значением функции приспособленности, поэтому именно он и был выбран в разработанной программе.

## Оптимальность расписания

Для того чтобы найти наиболее оптимальное расписание, используется фитнесс-функция. Она способна оценивать оптимальность решения, исходя из заданных требований. То есть если для пользователя важно, чтобы в расписании не было «окон» между уроками, то более плотно укомплектованные расписания фитнесс-функция будет оценивать выше, нежели другие. Если для пользователя важно, чтобы в расписании «окон» не было у преподавателей, то соответствующие расписания будут оцениваться выше. Таким образом, в зависимости от того, какое свойство интересует пользователя, меняется фитнесс-функция, за счет чего обеспечивается гибкость подхода. Выбранный подход позволяет построить расписание, «оптимальное» с любой точки зрения, исходя из того, что пользователь понимает под оптимальностью. При реализации алгоритма фитнесс-функции были учтены пожелания диспетчера по расписанию одной из школ города Добрянки.

# Практическое решение задачи составления расписаний

Данная глава содержит реализацию системы:

1. обоснование выбора инструментов реализации;
2. описание структуры системы;
3. формирование начальной популяции;
4. описание функции оценки.

Приведена статистика работы генетического алгоритма. Также одной из задач, решаемой на данном этапе является апробация системы на расписании, которое содержит 15 классов, 29 учителей, 438 уроков.

## Функция оценки (фитнесс-функция)

Функция оценки представляет из себя следующий алгоритм:

1. проходим по каждому классу ;
2. внутри класса проходим по каждому уроку в расписании ;
3. если встретилось окно в середине дня из оценки (*R*) вычитается 30 единиц, иначе за каждый урок начисляется единица;
4. если чередуются естественно-математические и гуманитарные предметы, то так же начисляется единица рейтинга;
5. оценка по каждому классу суммируются в общую оценку расписания.

Блок-схема данного алгоритма изображена на рисунке Рисунок 10.

Аналогично проходим по расписанию учителей и проверяем наличие окон у них, а также их предпочтения.

При случайной генерации особей мы получаем особь, имеющую оценку выше 400. Средняя оценка популяции около 200-300. При выполнении генетического алгоритма средняя оценка популяции растет, а разнообразие падает, и мы упираемся в максимум фитнесс-функции. При этом расписание получается без окон, как мы и задавали в фитнесс-функции. Если описать математически остальные требования, генетический алгоритм должен еще сильнее улучшить расписание для удовлетворения новых требований. Пример расписания представлен ниже в этой главе.

Цикл по классам

Цикл по урокам

R += 1

Встретилось окно

R −= 30

Да

Нет

Предметы чередуются

R += 1

Да

Нет

Конец

Начало

Оценка расписания R = 0

Рисунок – Блок-схема работы алгоритма оценки расписания

## Формирование начальной популяции

Генерация начального поколения – 100 особей. Не все начальные особи позволяют получить раскраски в 30 цветов, но методом экспериментов получилось, что 10% особей начальной популяции раскрашиваются успешно в 30 цветов. При условии, что мы создаем расписание для 15 классов – по 3 класса в параллели. Даже если в начальной популяции не было особи, которая раскрашивается в 30 цветов, в процессе генетического алгоритма очень велика вероятность что такая появится. Особи, которые нельзя корректно уложить в расписание штрафуются функцией оценки.

## Инструменты реализации

Языком программирования был выбран язык C#, так как объектно-ориентированная парадигма отлично ложится на поставленную задачу, и особи удобно представляются в виде объектов для выполнения дальнейших операций с ними.

Для реализации программной системы была выбрана среда Microsoft Visual Studio 2017, которая имеет удобный интерфейс и поддержку многих языков программирования, простоту освоения и множество инструментов для улучшения кода.

## Структура программы

Диаграмма понятий рассматриваемой предметной области приведена на рисунке Рисунок 11, а диаграмма классов – на рисунке Рисунок 12.

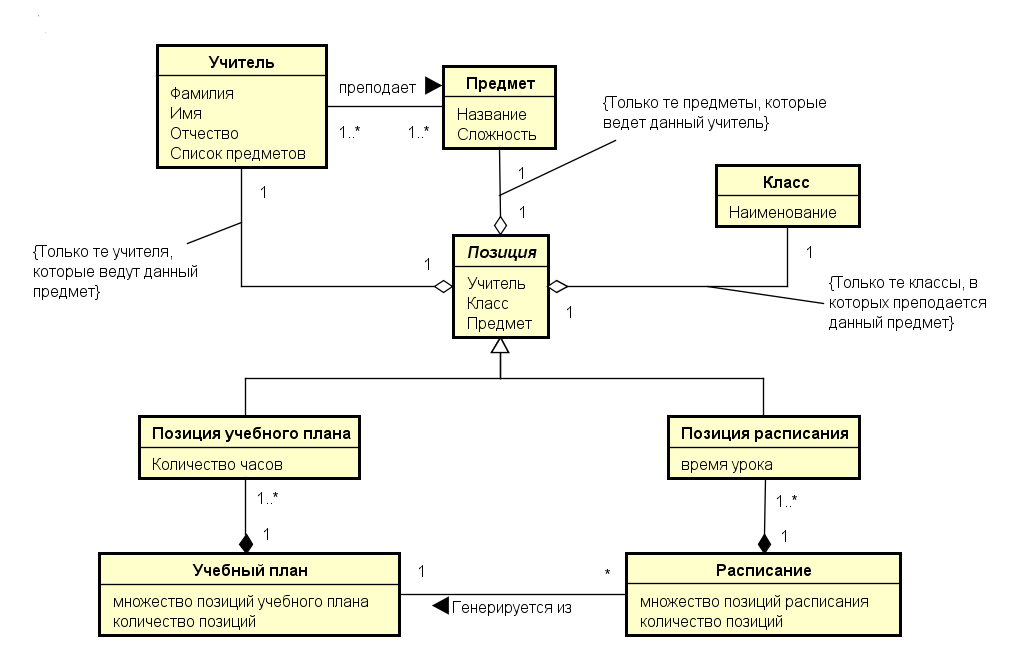


Рисунок – Диаграмма понятий

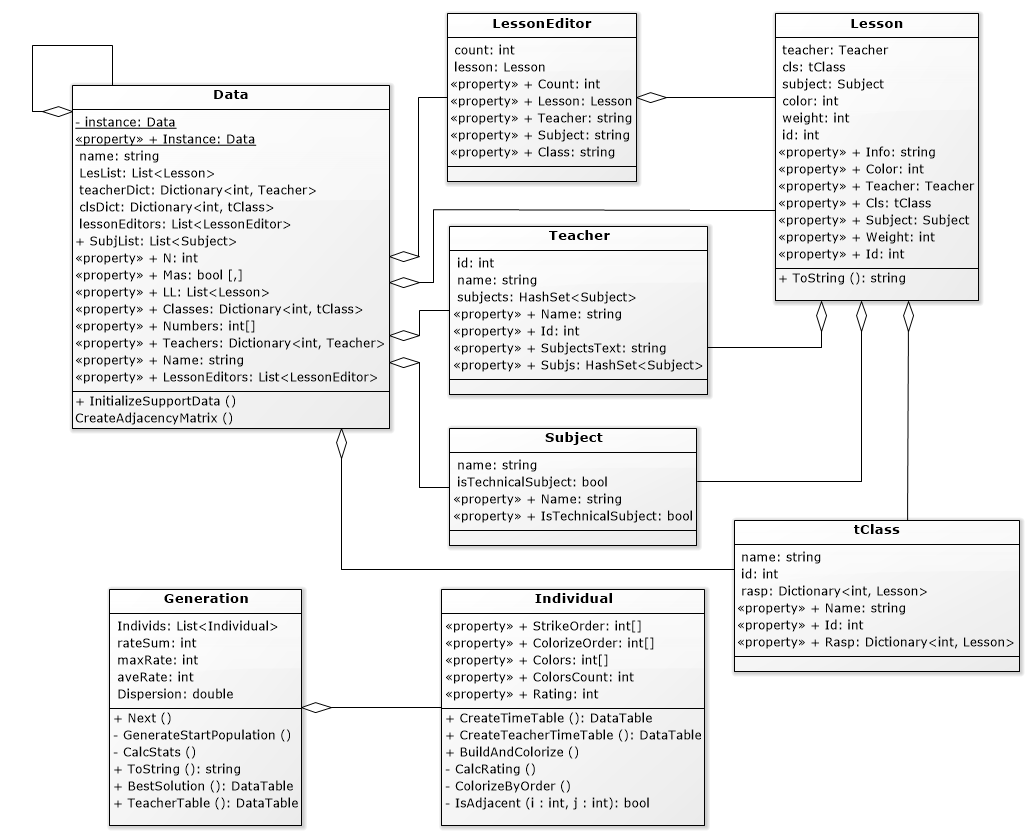


Рисунок - Диаграмма классов

В разрабатываемой программной системе содержится:

1. класс *Teacher* – хранит в себе сведения о преподавателе, его ФИО, список предметов, которые он преподает;
2. класс *tClass* – для хранения информации о классе, наименование класса, различная дополнительная информация;
3. класс *Subject* – представляет из себя школьный предмет;
4. класс *Lesson* – хранит информацию о сочетаниях преподаватель, предмет и класс. Представляет из себя классический урок. В программе играет роль вершины графа;
5. класс *Data* – агрегирует в себе объекты вышеперечисленных классов и выполняет основные операции над ними, такие как чтение, запись информации о преподавателях и классах;
6. класс *WorkWithExcel* – отвечает за работу с Экселем. А конкретно, вывод полученного расписания;
7. класс *Rand* – реализует потокобезопасный генератор случайных чисел. И функцию для «перемешивания» элементов списка;
8. класс *Individual* – представляет из себя особь генетического алгоритма. Хранит в себе информацию о генотипе, фенотипе. Реализует функции по переводу генотипа в фенотип и обратно, раскраске графа, укладка раскрашенного графа в расписание, оценки полученного расписания;
9. класс Generation – множество особей, называемых популяцией, хранит список особей и реализует функцию генерации начального поколения и перехода между поколениями.

## Интерфейс пользователя

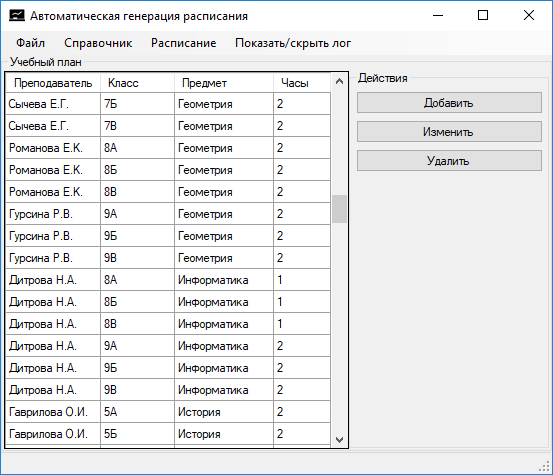
Главная форма программы изображена на рисунке Рисунок 13, на ней находится список позиций учебного плана.

Рисунок 13 – Главная форма программы

Система должна позволять пользователю редактировать учебный план, что подразумевает добавление, изменение и удаление позиций учебного плана (см. рисунок Рисунок 14).

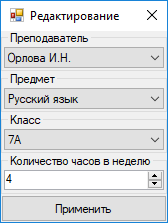


Рисунок 14 – Форма добавления/изменения позиции учебного плана

Меню программы содержит следующие пункты:

1. Файл (открытие и сохранение файла, экспорт расписания в Excel)
2. Справочник (открывает окна редактирования списка преподавателей, классов, предметов)
3. Расписание (настройка параметров генерации, генерации расписания, открытие расписания для учеников/учителей)
4. Показать/скрыть лог (при нажатии отображает статистическую информацию во время генерации расписания)

Форма настройки параметров генерации позволяет выбрать диспетчеру по расписанию параметры генерации расписания, это отражено на рисунке Рисунок 15.

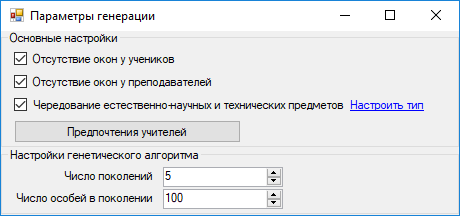


Рисунок 15 – Форма настроек параметров генерации

Форма отображения расписания показана на рисунке Рисунок 16. Она отображает результат работы программы, который может быть экспортирован в Excel.

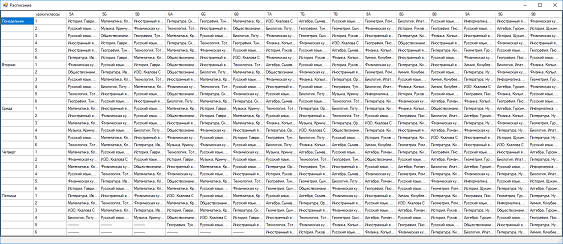


Рисунок 16 – Форма отображения расписания

## Статистика работы генетического алгоритма

Система была протестирована на реальных данных: 15 классов, 29 учителей, 438 уроков. Средняя оценка начальной популяции составляла около 300, что отражено на рисунке Рисунок 17. При выполнении генетического алгоритма средняя оценка популяции растет, а разнообразие падает, что отражено на рисунке Рисунок 19, и мы достигаем локального максимума функции.

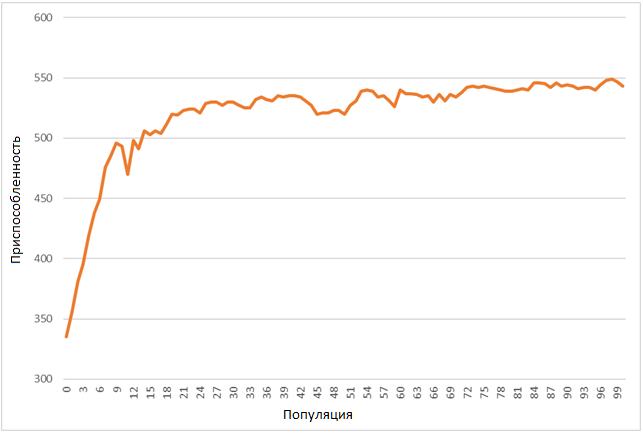


Рисунок 17 – График изменения среднего значения фитнесс-функции

Динамика работы ГА отражена на рисунке Рисунок 18, который демонстрирует высокую скорость и монотонный характер роста функции приспособленности, что подтверждает корректность реализованного алгоритма.

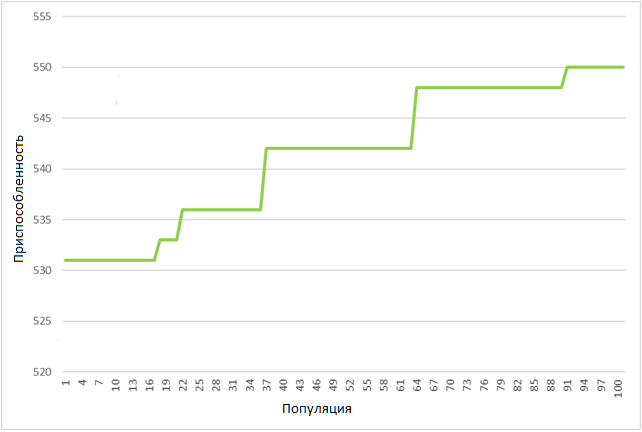


Рисунок 18 – График изменения максимального значения фитнесс-функции

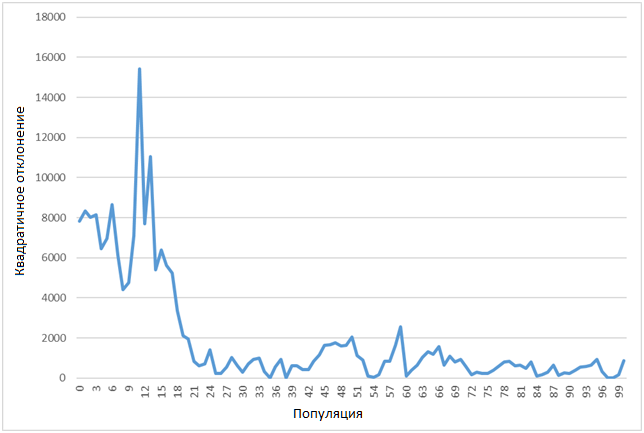


Рисунок 19 – График изменения разнообразия особей в популяции

При использовании случайного способа генерации расписания для четырех классов одной параллели лишь 1% из 1000 сгенерированных расписаний получался корректным. А при выбранном способе кодирования уже при начальной генерации 30% особей были корректными.

## Результат работы генетического алгоритма

Фрагмент корректно сгенерированного расписания.

Таблица 5 - Фрагмент расписания

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | 5А | 5Б | 5В |
| понедельник | 1 | География  Тунчак С.К. | Технология  Тоттава Н.И. | Биология  Лотуш Е.М. |
| 2 | Математика Климов М.Т. | Физкультура  Мордлина Г.А. | География  Тунчак С.К. |
| 3 | Русс/литер  Иванов В.В. | Технология  Тоттава Н.И. | Музыка  Кринчук М.В. |
| 4 | Иностранный яз.  Петров К.С. | Математика  Климов М.Т. | Русс/литер  Иванов В.В. |
| 5 | Математика Климов М.Т. | Иностранный яз. Петров К.С. | История/общество.  Гаврилова О.И. |
| 6 | ИЗО  Окулова С.С. | Русс/литер  Иванов В.В. | Математика  Климов М.Т. |

Как видно из таблицы Таблица 5 получаемое в результате работы ГА расписание не содержит «окон», что по сути и отражено экспертами в алгоритме оценивания оптимальности расписания как главный критерий.

Полное расписание представлено в приложении А.

# Заключение

Апробация работы была проведена на Всероссийской научно-практической конференции молодых ученых с международным участием «Математика и междисциплинарные исследования – 2018» [13].

Во время выполнения работы были решены следующие задачи:

1. выполнен обзор 8 источников литературы по данному вопросу и обзор 3 подходов к решению задачи;
2. произведен обзор 4 программ-аналогов по составлению расписания;
3. изучены нормативные документы по составлению школьного расписания, на основе которых сформулированы требования к задаче;
4. определена структура для хранения расписания;
5. разработан генетический алгоритм, особенностью которого является то, что он позволяет получить наиболее оптимальное расписание в соответствии с фитнесс-функцией, которая является настраиваемой;
6. разработана программа, позволяющая автоматически составлять расписание в соответствии с введенным учебным планом, установленными ограничениями и предпочтениями со стороны преподавателей;
7. произведен анализ результатов, которые показали высокую скорость и монотонный характер роста функции приспособленности, что подтверждает корректность реализованного алгоритма;
8. учтены пожелания диспетчера по расписанию.

Перспективы дальнейшего развития:

1. деление классов на группы для таких предметов как информатика, иностранный язык;
2. разработка интерфейса для изменения фитнесс-функции, чтобы не прибегать к переписыванию кода, при адаптации программы к конкретному учебному заведению.

Таким образом, задачи были решены, цель, поставленная в работе, была достигнута полностью.

Возможность адаптации системы продемонстрирована на реальных данных: 15 классов, 29 учителей, 438 уроков. Программный код содержит 1329 строк. Исполняемый файл имеет размер 10 КБ;

С учётом вышесказанного, систему можно будет использовать в школах для облегчения процесса составления расписания.

# Список литературы

|  |  |
| --- | --- |
| 1. | Mitchell M. An Introduction to Genetic Algorithms. Cambridge: The MIT Press, 1996. |
| 2. | ХроноГраф 3.0 Мастер [Электронный ресурс] // Хронобус: [сайт]. URL: http:/​/​www.chronobus.ru/​normbase/​detail.php?ELEMENT\_ID=1772564 (дата обращения: 27.05.2018). |
| 3. | 1С:Автоматизированное составление расписания. Школа [Электронный ресурс] // Отраслевые и специализированные решения 1С:Предприятие: [сайт]. URL: https:/​/​solutions.1c.ru/​catalog/​timetable/​features (дата обращения: 20.05.2018). |
| 4. | АВТОРасписание [Электронный ресурс] // : Лаборатория ММиИС : [сайт]. URL: https:/​/​www.mmis.ru/​programs/​avtor (дата обращения: 20.05.2018). |
| 5. | Gaspero L.D., McCollum B., Schaerf A. The Second International Timetabling Competition 2007 [Электронный ресурс] // Second International Timetabling Competition: [сайт]. URL: http:/​/​www.cs.qub.ac.uk/​itc2007/​curriculmcourse/​report/​curric ulumtechreport.pdf (дата обращения: 20.05.2018). |
| 6. | СанПиН 2.4.2.2821-10 -2015 Санитарно-эпидемиологические требования к условиям. введ. 2016. |
| 7. | Holland J.H. Adaptation in Natural and Artificial Systems. MIT Press Cambridge, 1975. |
| 8. | Чуприна С.И., Басов Д.В. Подход к разработке интегрированной инструментальной среды составления учебных расписаний // Актуальные проблемы математики, механики, информатики: материалы Международной научно-методической конференции, посвященной 90-летию высшего математического образования на Урале. Пермь. 2006. С. 163-165. |
| 9. | Soraya R. Examining the Role of Local Optima and Schema Processing in Genetic Search. 1999. |
| 10. | Whitley D. A Genetic Algorithm Tutorial. Kluwer Academic Publishers, 1993. |
| 11. | Курейчик В.М. Генетические алгоритмы. Физматлит, 2006. |
| 12. | Blickle T., Thiele L. A Comparison of Selection Schemes used in Genetic Algorithm. MIT Press Cambridge, 1995. |
| 13. | Макурин Р.А., Городилов А.Ю. Автоматическое составление учебных расписаний // Математика и междисциплинарные исследования – 2018. гл. ред. А. П. Шкарапута. - Пермь. 2018. С. 64-66. |

x

# Приложение А. Сгенерированное расписание







# Приложение Б. Листинг основных классов программы

public static class Data

{

static List<Lesson> LesList;

static Dictionary<int, Teacher> teacherDict;

public static Dictionary<int, tClass> clsDict;

public static int N { get; set; }

public static bool [,] Mas { get; set; }

public static List<Lesson> LL

{

get { return LesList; }

set { LesList = value; }

}

public static Dictionary<int, tClass> Classes

{

get { return clsDict; }

set { clsDict = value; }

}

public static int[] Numbers { get; private set; }

public static void ReadXML()

{

ReadTeachers();

ReadClasses();

ReadLessons();

CreateAdjacencyMatrix();

Numbers = new int[N];

for (int i = 0; i < N; i++)

{

Numbers[i] = i;

}

}

public static bool isMathSubj(string name)

{

return name == "Математика" || name == "Информатика" || name == "Физика" || name == "География" || name == "Химия" || name == "Биология";

}

static void ReadTeachers()

{

XmlSerializer teacherSerializer = new XmlSerializer(typeof(List<Teacher>), new XmlRootAttribute() { ElementName = "Teachers" });

TextReader teachReader = new StreamReader("teachers.xml");

List<Teacher> tList = (List<Teacher>)teacherSerializer.Deserialize(teachReader);

teachReader.Close();

teacherDict = tList.ToDictionary(x => x.Id, x => x);

}

static void ReadClasses()

{

XmlSerializer clsSerializer = new XmlSerializer(typeof(List<tClass>), new XmlRootAttribute() { ElementName = "Classes" });

TextReader clsReader = new StreamReader("classes.xml");

List<tClass> clsList = (List<tClass>)clsSerializer.Deserialize(clsReader);

clsReader.Close();

clsDict = clsList.ToDictionary(x => x.Id, x => x);

}

static void ReadLessons()

{

List<LessonHelp> lesHelp;

LesList = new List<Lesson>();

XmlSerializer lessonSerializer = new XmlSerializer(typeof(List<LessonHelp>), new XmlRootAttribute() { ElementName = "Lessons" });

TextReader lesReader = new StreamReader("lessons.xml");

lesHelp = (List<LessonHelp>)lessonSerializer.Deserialize(lesReader);

lesReader.Close();

N = lesHelp.Count;

for(int i = 0; i < N; i++)

LesList.Add(new Lesson(teacherDict[lesHelp[i].TeacherId], clsDict[lesHelp[i].ClsId], i));

}

static void CreateAdjacencyMatrix()

{

Mas = new bool[N, N];

for (int i = 0; i < N; i++)

{

for (int j = i + 1; j < N; j++)

{

if (LesList[i].Teacher == LesList[j].Teacher || LesList[i].Cls == LesList[j].Cls)

{

Mas[i, j] = Mas[j, i] = true;

LesList[i].Weight++;

LesList[j].Weight++;

}

}

}

}

}

class Generation

{

List<Individual> Individs;

int rateSum;

int maxRate;

int aveRate;

double Dispersion;

public Generation()

{

Individs = new List<Individual>(Properties.Settings.Default.PopulationCount);

GenerateStartPopulation();

CalcStats();

}

public void Next()

{

int randomCount = Individs.Count \* 20 / 100;

int childsCount = Individs.Count \* 60 / 100;

int lastCount = Individs.Count - randomCount - childsCount;

Rand.Shuffle(Individs);

List<Individual> childs = new List<Individual>(childsCount);

List<Individual> random = new List<Individual>(Individs.Skip(Math.Max(0, Individs.Count - randomCount)));

int index, firstRnd, firstParent, secondRnd, secondParent, curSum;

for (int i = 0; i < 30; i++)

{

index = curSum = firstParent = secondParent = 0;

firstRnd = ThreadSafeRandom.ThisThreadsRandom.Next(rateSum);

secondRnd = ThreadSafeRandom.ThisThreadsRandom.Next(rateSum);

while(index + 1 < Individs.Count && (curSum < firstRnd || curSum < secondRnd))

{

if (curSum < firstRnd)

firstParent++;

if (curSum < secondRnd)

secondParent++;

curSum += Individs[index++].Rating;

}

firstParent = firstParent > Individs.Count ? Individs.Count - 1 : firstParent;

secondParent = secondParent > Individs.Count ? Individs.Count - 1 : firstParent;

childs.Add(new Individual(Individs[firstParent], Individs[secondParent]));

childs.Add(new Individual(Individs[secondParent], Individs[firstParent]));

}

Individs.Sort(delegate (Individual x, Individual y)

{

return -x.Rating.CompareTo(y.Rating);

});

Parallel.ForEach(childs, (curChild) => curChild.BuildAndColorize());

Individs.RemoveRange(lastCount, Individs.Count - lastCount);

Individs.AddRange(childs);

Individs.AddRange(random);

CalcStats();

}

private void GenerateStartPopulation()

{

Data.ReadXML();

int startCount = Properties.Settings.Default.PopulationCount;

for (int i = 0; i < startCount; i++)

Individs.Add(new Individual());

Parallel.ForEach(Individs, (curIndivid) => curIndivid.BuildAndColorize());

}

private void CalcStats()

{

Dispersion = rateSum = aveRate = maxRate = 0;

foreach (Individual indiv in Individs)

{

maxRate = indiv.Rating > maxRate ? indiv.Rating : maxRate;

rateSum += indiv.Rating;

}

aveRate = rateSum/Individs.Count;

foreach (Individual indiv in Individs)

{

Dispersion += Math.Pow((indiv.Rating - aveRate), 2);

}

Dispersion /= Individs.Count;

}

public override string ToString()

{

return String.Format("maxRate: {0}, aveRate: {1}, dispersion: {2, -8} \n", maxRate, aveRate, Dispersion);

}

public void BestSolution()

{

Individs.Max().PrintTimeTable();

}

}

class Individual : IComparable<Individual>

{

public int[] StrikeOrder { get; private set; }

public int[] ColorizeOrder { get; private set; }

public int[] Colors { get; private set; }

public int ColorsCount { get; private set; }

public int Rating { get; private set; }

/// <summary>

/// Конструктор для случайного порядка

/// </summary>

public Individual()

{

Colors = new int[Data.N];

StrikeOrder = new int[Data.N];

ColorizeOrder = new int[Data.N];

Array.Copy(Data.Numbers, ColorizeOrder, Data.N);

Rand.Shuffle(ColorizeOrder);

Encode();

}

/// <summary>

/// Конструктор для создания потомка

/// </summary>

/// <param name="indA">Первая особь родитель</param>

/// <param name="indB">Вторая особь родитель</param>

public Individual(Individual indA, Individual indB)

{

Colors = new int[Data.N];

StrikeOrder = new int[Data.N];

ColorizeOrder = new int[Data.N];

int middle = Data.N / 2;

int i;

for (i = 0; i < middle; i++)

StrikeOrder[i] = indA.StrikeOrder[i];

for (i = middle; i < Data.N; i++)

StrikeOrder[i] = indB.StrikeOrder[i];

Decode();

if (ThreadSafeRandom.ThisThreadsRandom.Next(100) > 95)

Mutate();

}

public void PrintTimeTable()

{

int clsCount = Data.clsDict.Count;

Dictionary<int, Dictionary<int, Lesson>> TimeTable = new Dictionary<int, Dictionary<int, Lesson>>(clsCount);

foreach (int key in Data.clsDict.Keys)

TimeTable.Add(key, new Dictionary<int, Lesson>());

// подготовка таблицы для расписания ^^^

for (int i = 0; i < Data.N; i++)

TimeTable[Data.LL[i].Cls.Id].Add(Colors[i], Data.LL[i]);

DataTable dt = new DataTable();

dt.Columns.Add("уроки\\классы");

string[] tmpList = new string[31];

for (int i = 1; i < 31; i++)

{

dt.Columns.Add(i.ToString());

}

Lesson curLes;

foreach (Dictionary<int, Lesson> clsTimeTable in TimeTable.Values)

{

tmpList[0] = clsTimeTable.First().Value.Cls.Name;

for (int i = 1; i < 31; i++)

{

curLes = null;

clsTimeTable.TryGetValue(i, out curLes);

tmpList[i] = curLes?.ToString() ?? "-----------";

}

dt.Rows.Add(tmpList);

}

WorkWithExcel.ExportToExcel(WorkWithExcel.GenerateTransposedTable(dt));

}

public void BuildAndColorize()

{

ColorizeByOrder();

CalcRating();

}

private void CalcRating()

{

int clsCount = Data.clsDict.Count;

Dictionary<int, Dictionary<int, Lesson>> TimeTable = new Dictionary<int, Dictionary<int, Lesson>>(clsCount);

foreach(int key in Data.clsDict.Keys)

TimeTable.Add(key, new Dictionary<int, Lesson>());

// подготовка таблицы для расписания ^^^

for (int i = 0; i < Data.N; i++)

TimeTable[Data.LL[i].Cls.Id].Add(Colors[i], Data.LL[i]);

// подсчет рейтинга

Rating = 0;

if (ColorsCount > 29) // 34

Rating = -100;

Lesson curLesson;

bool lessonsBefore = false, emptyBefore = false, isPrevMath, curMath = false;

foreach (Dictionary<int, Lesson> clsTimeTable in TimeTable.Values)

{

for (int i = 1; i < 31; i++) // 36

{

if ((i - 1) % 6 == 0) // 7

lessonsBefore = emptyBefore = false;

curLesson = null;

clsTimeTable.TryGetValue(i, out curLesson);

if (curLesson == null)

emptyBefore = lessonsBefore;

else

{

Rating += emptyBefore ? -30 : 1;

isPrevMath = curMath;

curMath = Data.isMathSubj(curLesson.Teacher.Subjects);

if (lessonsBefore)

Rating += isPrevMath != !curMath ? 1 : 0;

lessonsBefore = true;

}

}

}

}

private void ColorizeByOrder()

{

ColorsCount = 0;

for (int i = 0; i < Data.N; i++)

{

Colors[ColorizeOrder[i]] = 1;

HashSet<int> hashSet = new HashSet<int>();

for (int j = 0; j < i; j++)

{

if (i != j && IsAdjacent(i, j))

{

hashSet.Add(Colors[ColorizeOrder[j]]);

while (hashSet.Contains(Colors[ColorizeOrder[i]]))

Colors[ColorizeOrder[i]]++;

if (ColorsCount < Colors[ColorizeOrder[i]])

ColorsCount = Colors[ColorizeOrder[i]];

}

}

}

}

private bool IsAdjacent(int i, int j)

{

return Data.Mas[Data.LL[ColorizeOrder[i]].Id, Data.LL[ColorizeOrder[j]].Id];

}

private void Encode() // закодировать - приведение к виду, удобному для скрещивания

{

List<int> numbers = new List<int>(Data.Numbers);

int curDelete;

for (int i = 0; i < Data.N; i++)

{

curDelete = numbers.FindIndex(x => x == ColorizeOrder[i]);

numbers.RemoveAt(curDelete);

StrikeOrder[i] = curDelete;

}

}

private void Decode() // раскодировать - приведение к привычному виду порядка раскраски

{

List<int> numbers = new List<int>(Data.Numbers);

for (int i = 0; i < Data.N; i++)

{

ColorizeOrder[i] = numbers[StrikeOrder[i]];

numbers.RemoveAt(StrikeOrder[i]);

}

}

/// <summary>

/// Мутация

/// </summary>

/// <param name="count"> количество мутирующих генов </param>

private void Mutate(int count = 2)

{

int firstIndex, secondIndex, tmpValue;

for(int i = 0; i < count; i++)

{

firstIndex = ThreadSafeRandom.ThisThreadsRandom.Next(Data.N);

secondIndex = ThreadSafeRandom.ThisThreadsRandom.Next(Data.N);

tmpValue = ColorizeOrder[firstIndex];

ColorizeOrder[firstIndex] = ColorizeOrder[secondIndex];

ColorizeOrder[secondIndex] = tmpValue;

}

Encode();

}

int IComparable<Individual>.CompareTo(Individual other)

{

return Rating.CompareTo(other.Rating);

}

}