Национальный технический университет Украины
«Киевский политехнический институт»
Факультет информатики и вычислительной техники
Кафедра вычислительной техники

Курсовой проект

по дисциплине

«Операционные системы»

Руководитель

_____ (Симоненко В. П.)

Допущен к защите

_____

Защищен с оценкой

_____

Выполнил

студент ФИВТ, группы ИВ-73

Захожий И. А.

Киев-2011

№ строки	Формат	Обозначения	Наименование	Кол. лист.	№ экз.	Прим.
1						
2			Документация общая			
3						
4			Заново разработанная			
5						
6	А4	ИАЛЦ.462637.001 ВП	Ведомость проекта	1		
7						
8	А4	ИАЛЦ.462637.002 ТЗ	Техническое задание	4		
9						
10	А4	ИАЛЦ.462637.003 ПЗ	Пояснительная записка	21		
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						

Подп. и дата | Инв.№ дубл. | Взам. инв. № | Подп. и дата | Инв.№ подл.

Изм.	Лист	№ документа	Подп.	Дата
Разраб		Захожий И.А.		
Провер.		Симоненко В.П.		
Н.контр.				
Утв.				

ИАЛЦ.462637.001 ВП

Ведомость проекта

Лит.	Лист	Листов
Т	1	1

НТУУ «КПИ», ФИВТ, группа ИВ-73

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Содержание

1. Наименование и область применения 2
2. Основание для разработки 2
3. Цель и назначение разработки 2
4. Требования к программе 2
   4.1 Требования к программной модели 2
   4.2 Требования к надежности 3
   4.3 Требования к составу и порядку технических средств 3
   4.4 Требования к программной совместимости 3
5. Требования к документации 3
6. Этапы разработки 4

1. Наименование и область применения

В данном курсовом проекте разработан алгоритм планирования для вычислительной системы с топологией «бинарное дерево». Алгоритм был разработан с целью реализации данной топологии программными средствами для погружения заданного пользователем ациклического графа задачи на процессоры вычислительной системы.

2. Основание для разработки

Основанием для разработки служит задание на курсовой проект по курсу «Операционные системы».

3. Цель и назначение разработки

Целью разработки данного программного продукта является закрепление умений и навыков в программировании на языках высокого уровня, а также закрепление знаний о параллельных вычислительных системах, алгоритмах планирования, полученных при изучении курса "Операционные системы".

4. Требования к программе

4.1 Требования к программной модели

Данная программная модель должна погружать ациклические направленные графы задач на вычислительную систему с топологией «бинарное дерево» по разработанному алгоритму.

4.2 Требования к надежности

Программа должна выполнять моделирование при любом количестве вершин графа задания и для любого количества процессоров.

Надёжность программного продукта должна быть проверена в результате тестирования на компьютерах разной производительности, на различных версиях операционной системы Microsoft Windows и с различными входными параметрами.

4.3 Требования к составу и порядку технических средств

Программа должна быть разработана на персональном компьютере на одной из языков высокого уровня. Оформить результаты работы в виде технической документации на проект, включающий техническое задание, пояснительную записку, необходимые ведомости в соответствии с ГСТУ.

4.4 Требования к программной совместимости

Программа должна работать в средах, совместимых с операционной системой Microsoft Windows.

5. Требования к документации

Курсовой проект должен включать следующие документы:

1. Техническое задание
2. Пояснительная записка
3. Приложение (листинг программы).

6. Этапы разработки

1. Согласование технического задания
2. Разработка алгоритмов и структуры данных
3. Разработка программного обеспечения
4. Тестирование программного обеспечения
5. Оформление документов
6. Сдача курсового проекта.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Содержание

Введение.....2

1. Обзор существующих решений.....4

2. Описание заданной топологии.....9

3. Алгоритм планирования.....10

4. Описание разработанной программы.....13

5. Экспериментальные результаты работы программы.....15

6. Заключение.....19

Список использованной литературы.....20

Подп. и дата							
Инв.№ дубл.							
Взам. инв. №							
Подп. и дата							
					ИАЛЦ. 462637.003 ПЗ		
	Изм.	Лист	№ документа	Подпись	Дата		
Инв.№ подл.	Разработал		Захожий И.А.			Лит.	Лист 1 / Листов 21
	Проверил		Симоненко В.П.			Пояснительная записка	НТУУ «КПИ», ФИВТ, группа ИВ-73
	Н. контр.						
	Утвердил						

Введение

В настоящее время для решения достаточно сложных задач не хватает мощности одиночных процессоров, несмотря на довольно большие величины, выражаемые в гигагерцах и сотнях мегафлопсов (мегафлопс — 1 миллион операций с плавающей запятой в секунду). Приведем несколько направлений, где необходима высокая мощность вычислительных систем:

- моделирование сложных химических или физических процессов
- векторно-матричные вычисления
- обработка 3D графики
- хранилища информации.

Существуют два пути повышения производительности вычислительных комплексов:

- повышение частотных возможностей элементной базы
- увеличение количества одновременно работающих вычислителей в системе.

Недостаток первого в том, что повышение частоты процессора ограничено размером кристалла и, судя по всему, производители уже подошли к границе использования данного метода. Так что для решения задач высокой сложности данный метод не может эффективно использоваться.

Второй путь приводит к необходимости разработки эффективного параллельного алгоритма задачи, а также задачи оптимального планирования и отображения параллельного алгоритма в структуру вычислительной системы.

По аппаратным особенностям, для решения задачи параллельного вычисления алгоритма более удобны распределенные системы.

Для эффективной работы параллельной программы на распределенной системе необходим адаптивный алгоритм, которому приходится решать NP-полную задачу планирования задач на ресурсы системы. В общем виде

математическое решение для данных задач на современных компьютерах может занимать от нескольких десятков минут до нескольких месяцев. Поэтому для решения этих проблем используют эвристические подходы.

Учитывая, что исходная задача задана в виде ориентированного ациклического графа в ярусно-параллельной форме, их можно формализовать в три группы:

- *списочные*
- *кластерные*
- *генетические.*

Списочное планирование [2], [3], [4] эффективно только для систем с общей шиной или однородных полносвязных систем. Работа алгоритма заключается в формировании списка очереди готовых вычислительных заданий с помощью одного из эвристических методов и затем оптимального, имея в виду пересылки данных, назначения очереди вычислительных работ на свободные процессоры. Две эти задачи решаются независимо одна от другой, что приводит к неэффективности вычислений.

Кластерное планирование [5], [6], [7] в чистом виде применяется только для систем, где нет ограничений на выделяемые ресурсы, или в масштабируемых вычислительных системах. Задания из исходного графа делятся на группы (кластеры). Этот процесс называется кластеризацией, и его решение в общем виде имеет экспоненциальную временную сложность. Задания одной группы (кластера) выполняются на одном процессоре.

Особенность генетического планирования [8], [9] заключается в возможности решения задач планирования и назначения нераздельно одна от другой, это приводит к уменьшению сетевого трафика и оптимальной загрузки процессоров в системе. В результате эффективность использования процессоров выше, а время решения задачи приближается к критическому (идеальному без учета пересылок).

1. Обзор существующих решений

В работах [10], [11] показано, что почти все задачи построения расписаний являются NP-полными. NP-полнота задач построения расписаний обусловила широкое применение для их решения эвристических алгоритмов, основанных на жадных стратегиях, и итерационных алгоритмов: генетические и эволюционные алгоритмы, алгоритмы имитации отжига, алгоритмы случайного поиска (ненаправленного, направленного, направленного с самообучением), алгоритмы детерминированной коррекции расписания.

Алгоритмы, основанные на жадных стратегиях, подразумевают декомпозицию задачи на подзадачи (вложении задачи в семейство более простых задач). Задача построения расписания может быть разбита на подзадачи следующими способами:

1. Подзадача заключается в распределении i-го рабочего интервала в расписание, содержащее (i-1) рабочий интервал. Для решения задачи полного составления расписания НР требуется решить N таких подзадач, где N – число рабочих интервалов в программе. В каждой подзадаче число возможных вариантов ее решения равно M0, где M0<N – число возможных мест размещения i-го рабочего интервала.

2. Строится максимально параллельное расписание – время инициализации каждого рабочего интервала определяется готовностью его к выполнению (выполнены все предшественники). Подзадача заключается в ликвидации одного процессора и распределении его процессов по оставшимся процессорам. Для решения задачи полного построения расписания НР требуется решить M′-M таких подзадач, где M – число процессоров в ВС, для которой строится расписание, M′ – число процессоров для максимально параллельного расписания. Каждая подзадача заключается в решении NS- следующих подзадач: распределение рабочего интервала, принадлежащего ликвидируемому процессору, в один из

оставшихся процессоров, где NS- — число рабочих интервалов в ликвидируемом процессоре. Всегда справедливо NS-<N. Число возможных вариантов решения подзадачи распределения рабочего интервала всегда меньше N.

То есть, при данных способах разбиения задачи составления расписаний на подзадачи, оптимальное решение каждой подзадачи может быть получено за полиномиальное число шагов, например, полным перебором возможных вариантов решения. Число подзадач также полиномиально. Жадные алгоритмы, использующие второй способ разбиения задачи, могут иметь более высокую вычислительную сложность по сравнению с алгоритмами, использующими первый способ (верхняя оценка сложности алгоритмов соответственно равна O(N2) и O(N3-N2·M)), но они применимы и в случае, когда процесс может иметь более одного рабочего интервала и на расписание наложено ограничение "все рабочие интервалы одного процесса должны быть назначены на один и тот же процессор".

Свойство 1. Для жадных алгоритмов решения NP-полных задач составления расписаний не существует единой для всех подзадач локальной целевой функции, приводящей к наилучшему варианту расписания, получаемого алгоритмом.

Свойство 2. Для жадных алгоритмов решения NP-полных задач составления расписаний влияние на качество расписания используемых локальных целевых функций возрастает с ростом сложности функции T=f(HP,HW) по учитываемым временным задержкам.

Свойство 3. Для жадных алгоритмов решения NP-полных задач составления расписаний критерии в локальной целевой функции, определяющие привязку рабочих интервалов к процессорам, оказывают более сильное влияние, на качество, получаемого алгоритмом расписания, чем критерии, определяющие порядок рабочих интервалов на процессорах.

В работе [10] отмечена подверженность жадных алгоритмов построения расписаний аномалиям: улучшение параметров ВС и программы,

для которой составляется расписание, может приводить к увеличению времени выполнения расписания. В [12] приведены примеры, когда списочный алгоритм получает худшее по времени выполнения расписание при: 1)увеличении числа процессоров 2)уменьшении длительности выполнения рабочих интервалов 3)снятие ограничений на порядок следования рабочих интервалов 4)уменьшение длительности внутренних простоев процессоров. По крайне мере, одной из причин подверженности жадных алгоритмов аномалиями может быть свойство 1.

Итерационные алгоритмы можно разбить на два подкласса:

- алгоритмы, опирающиеся на метод проб и ошибок: генетические и эволюционные алгоритмы [8], [9], алгоритмы имитации отжига [13], алгоритмы случайного поиска (ненаправленного, направленного, направленного с самообучением) [14];
- алгоритмы детерминированной коррекции расписания.

Схематично работу этих алгоритмов для решения задачи построения расписания можно представить следующим образом:

1. Задать начальное приближение HP0, k=0.
2. Вычислить целевую функцию f(HPk,HW) и проверить выполнение ограничений (если в п.3 возможно получение $HP \notin HP^{*}_{1-5}$).
3. Получить HPk+1=D({HPi},{f(HPi,HW)},Prk:i∈(1,…,k)).
4. Если критерий останова не достигнут, то k=k+1 и перейти к п.2; в противном – завершить работу алгоритма.

Где, D — некоторая стратегия коррекции текущего расписания, Prk — параметры стратегии (для ряда стратегий, возможно, их изменение в ходе работы алгоритма).

Алгоритмы, опирающиеся на метод проб и ошибок, содержат элементы случайного выбора привязки и порядка, что дает возможность получать информацию о их влиянии на значения целевой функции и ограничений. Данная информация используется для адаптации алгоритма (в ходе его работы) к закону влияния привязки и порядка на значения

целевой функции и ограничений. Кроме возможности адаптации к решаемой задаче, алгоритмы, опирающиеся на метод проб и ошибок, наиболее полно удовлетворяют требованиям, к свойствам алгоритмов построения расписания при совместном проектировании аппаратных и программных средств ВС: 1)возможностям методов строить расписания для архитектур ВС с различным уровнем детализации; 2)универсальности методов в классе допустимых архитектур ВС; 3)совместимости методов построения расписания и методов выбора оптимальных параметров архитектуры.

Данные особенности алгоритмов обеспечивают эффективность их использования на системном уровне проектирования: определение числа процессоров, "типа" каждого процессора, топологии и характеристик компонентов (задержка, арбитраж и маршрутизация) коммутационной среды, распределение памяти (тип – ПЗУ/ОЗУ, объем, способ доступа – локальная/разделяемая, циклы считывания/записи) и построение расписания выполнения прикладной программы.

Алгоритмы детерминированной коррекции расписания для своего построения требуют некоторых априорных сведений о влиянии привязки и порядка на значения целевой функции и ограничений. Это обуславливает специализацию методов на конкретный класс архитектур ВС. Однако, для уточнения расписания (локальной оптимизации расписания) можно предложить алгоритмы детерминированной коррекции расписания исправляющие "плохие" фрагменты расписания и не использующие априорные сведения о влиянии привязки и порядка на значения целевой функции и ограничений. Если некоторый рабочий интервал находится длительное время в состояниях – ожидание данных от других рабочих интервалов или ожидание получения разделяемого ресурса, то можно попытаться уменьшить время нахождения рабочего интервала в этих состояниях путем изменения расположения в расписании рабочего интервала или его непосредственных предшественников. То есть, работа алгоритма основана на использование информации о "плохих" фрагментах

расписания, которая может быть определена при анализе динамики функционирования ВС. Такое построение алгоритмов детерминированной коррекции расписания позволяет устранить их узкую специализацию на конкретный класс архитектур ВС. Основная область эффективного применения алгоритмов детерминированной коррекции решений — уточнение расписаний на уровне регистровых передач при решении задачи уточнения деталей реализации ВС с учетом элементной базы и системного программного обеспечения.

В генетических алгоритмах система операций преобразования расписаний определена: операции скрещивания, мутации и селекции. Требуется выбрать такую форму представления расписаний, чтобы операции генетического алгоритма не приводили к получению недопустимых вариантов расписаний и любое допустимое расписание могло быть представлено в выбранной форме представления расписаний.

В алгоритмах имитации отжига, случайного поиска, детерминированной коррекции расписаний система операций преобразования расписаний вводится при разработке алгоритма. Для этих алгоритмов способ представления расписаний и система операций преобразования расписаний должными обладать следующими свойствами: применение операции должно приводить к допустимому варианту расписания и система операций должна быть функционально полной.

2. Описание заданной топологии

В данном курсовом проекте была разработана программа статического погружения графа заданий на топологию "бинарное дерево".

При использовании данной топологии система строится по схеме так называемого строго двоичного дерева, где каждый узел более высокого уровня связан с двумя узлами следующего по порядку более низкого уровня. Узел, находящийся на более высоком уровне, принято называть родительским, а два подключенных к нему нижерасположенных узла — дочерними. В свою очередь, каждый дочерний узел выступает в качестве родительского для двух узлов следующего более низкого уровня. Отметим, что каждый узел связан только с двумя дочерними и одним родительским. Пример топологии «бинарное дерево» для семи узлов представлен на рис. 2.1.

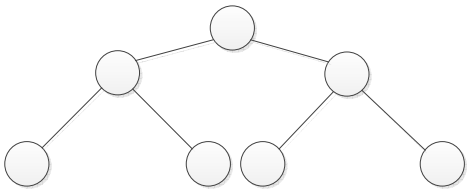


Рис. 2.1. Пример топологии «бинарное дерево» для семи узлов

Если h — высота дерева (количество уровней в древовидной системе), определяемая как $[\log_2 N]$, то такую систему можно охарактеризовать следующими параметрами: диаметр - D = 2(h — 1); порядок узла — d = 3; число узлов — l = N — 1; ширина бисекции — B = 1.

При больших объемах пересылок между несмежными узлами древовидная топология оказывается недостаточно эффективной, поскольку сообщения должны проходить через один или несколько промежуточных звеньев. Очевидно, что на более высоких уровнях системы вероятность затора из-за недостаточной высокой пропускной способности линий связи выше.

3. Алгоритм планирования

В данном курсовом проекте был разработан пространственный планировщик — программа, реализующая статическое планирование процессов для системы с топологией «бинарное дерево». Для этой цели был выбран генетический алгоритм.

Генетические алгоритмы являются одними из эволюционных алгоритмов, применяемых для поиска глобального экстремума функции многих переменных. Принцип их работы основан на моделировании некоторых механизмов популяционной генетики: манипулирование хромосомным набором при формировании генотипа новой биологической особи путем наследования участков хромосомных наборов родителей (кроссовер) и случайное изменение генотипа, известное в природе как мутация. Другим важным механизмом, заимствованным у природы, является процедура естественного отбора, направленная на улучшение от поколения к поколению приспособленности членов популяции путем большей способности к "выживанию" особей, обладающих определенными признаками.

Рассмотрим шаги генетического алгоритма погружения параллельной задачи в распределенную вычислительную систему.

1. Генерируется случайным образом N генотипов планирования графа. Информация о распределении задач по процессорам хранится в массивах. Индекс каждой ячейки массива — это номер вершины в графе задачи, а значение в ячейке — т номер процессора, на котором данная подзадача будет выполняться.

2. Строится матрица маршрутизации пакетов с данными внутри системы. Определяются кратчайшие пути между каждыми двумя процессорами.

3. Осуществляется равномерный кроссовер всех хромосом со случайным образом выбранной из списка с вероятностью 0,5.

4. Осуществляется мутация всех хромосом с вероятностью Pm = 0,05. Выполняется случайное изменение значений для поиска глобального экстремума.

5. Вычисляется целевая функция для всех генотипов:

а) сортируются все вершины по их приоритетам. Высший приоритет получает задача, которая находится на более высоком ярусе графа;

б) сортируются задачи по приоритетам пересылок данных из них. Приоритеты пересылок соответствуют приоритетам задач, для которых нужны вычисленные данные;

в) пока все вершины не выполнены, осуществляется:

- поиск невыполненных вершин, для которых получены все данные, и процессоры которых не были заняты при текущем событийном цикле. Время выполнения вершин равно их весу. Задачи выполняется потактово. Маскируем процессоры так, чтобы они снова не использовались для обработки новых задач при текущем цикле исполнения;
- на остальных уже занятых процессорах уменьшаются счетчики выполнения задач;
- если задача выполнена полностью, проверяется матрица связей графа задач. Если у данной задачи есть зависимые задачи, то пересылка добавляется в очередь связи, которая выбирается из таблицы маршрутизации. Время пересылки равно весу ребра графа задач. Пересылка начинается с следующего такта и производится потактово. Для этого связь маскируется до конца такта;
- для свободных связей между процессорами проверяются их очереди. Если они не пусты, то пересылки ставятся на выполнение;
- на занятых связях между процессорами уменьшаются счетчики выполнения пересылки;

- *если пересылка выполнена полностью, проверяется, сделана ли она в конечную точку. Если нет, то пересылка добавляется в очередь следующей связи, которая выбирается из таблицы маршрутизации и устанавливается флаг, чтобы не выполнить эту пересылку еще раз. Если да, то обнуляется соответствующее значение в матрице связей графа задач;*
- *увеличивается значение счетчика тактов на единицу;*

г) время выполнения алгоритма равно значению счетчика тактов.

6. Если это первая итерация, то сравниваются целевые функций, и M < N лучших структур заносим в отдельный массив. Они используются для того, чтобы решение в процессе итераций не ухудшилось. Если это не первая итерация, то лучшие структуры также участвуют в процессе сравнения.

7. Уничтожаются идентичные структуры, так как процесс выбора элитных структур приводит к созданию идентичных структур..

8. Структуры сортируются индексной сортировкой список структур и (N – M) последних худших структур удаляются.

9. Пока не закончилось число эпох (циклов алгоритма) или не достигнуто оптимальное для данной задачи решение, переходим на шаг 3.

4. Описание разработанной программы

В разработанной программе реализована возможность ввода и редактирования погружаемого графа в графическом виде. Главное окно программы (рис. 4.1) предоставляет доступ к инструментам редактирования.

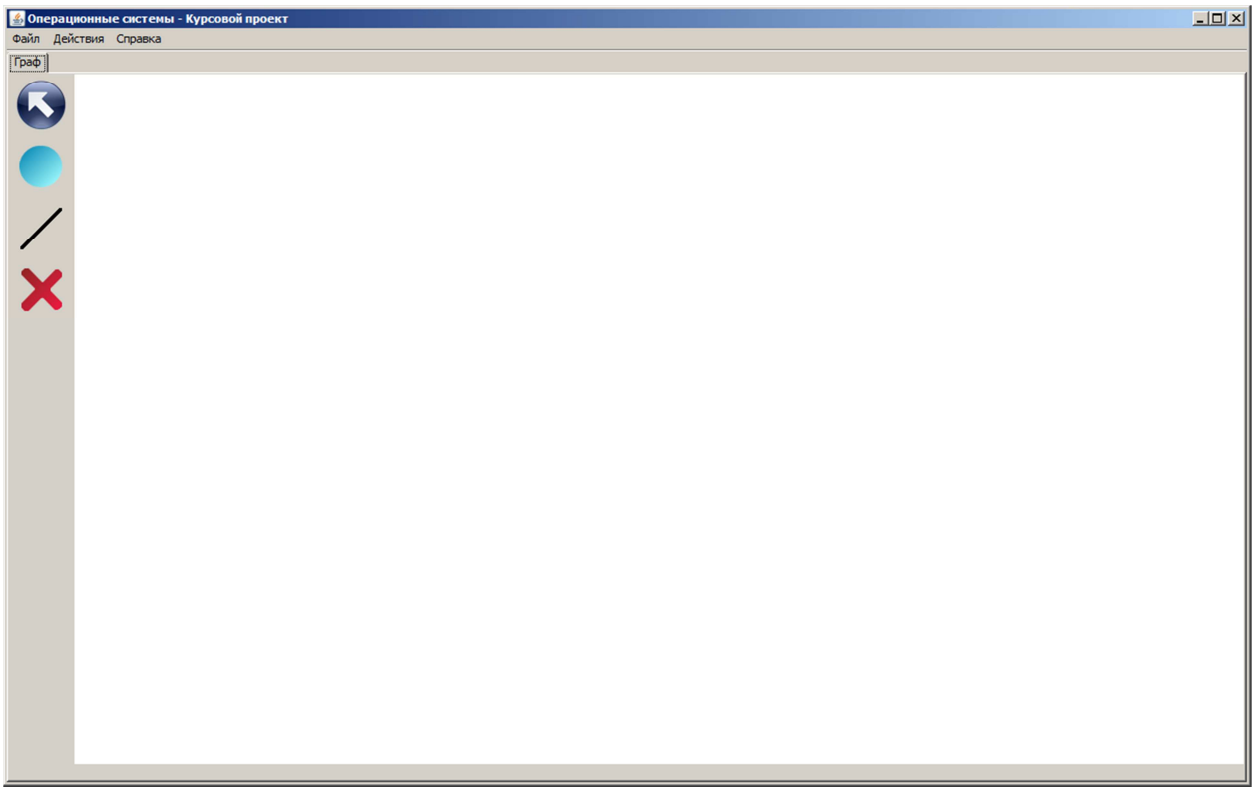


Рис. 4.1. Главное окно программы

Меню «Файл» содержит пункты: «Создать граф» (комбинация клавиш Ctrl-N), «Открыть граф...» (комбинация клавиш Ctrl-O), «Сохранить граф» (комбинация клавиш Ctrl-S), «Сохранить граф как...» (комбинация клавиш Ctrl-Shift-N), «Закрыть граф» (комбинация клавиш Ctrl-W). Они позволяют создать, закрыть, сохранить и восстановить погружаемый граф. Также меню «Файл» содержит пункт «Выход» (комбинация клавиш Ctrl-E) для закрытия программы.

Инструменты для ввода или редактирования графа находятся на панели в правой части главного окна программы. Кнопка «Перемещение вершин» позволяет изменить расположение вершины в рабочей области.

Кнопки «Добавление вершины» и «Добавление ребра» предназначены для добавления в граф новой вершины и нового ребра соответственно. При добавлении вершины или ребра вызывается диалоговое окно для ввода веса вершины или ребра (рис. 4.2). Кнопка «Удалить вершину» позволяет удалять вершины из графа. Для удаления ребра необходимо вызвать контекстное меню вершины и выбрать необходимое ребро.

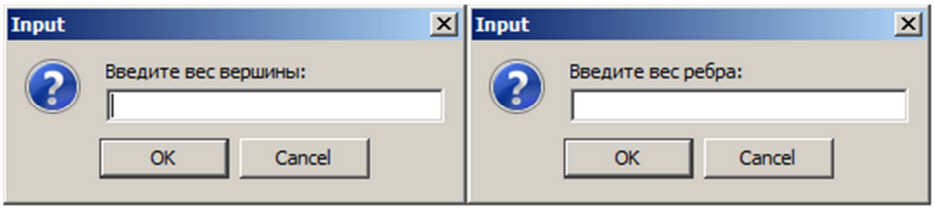
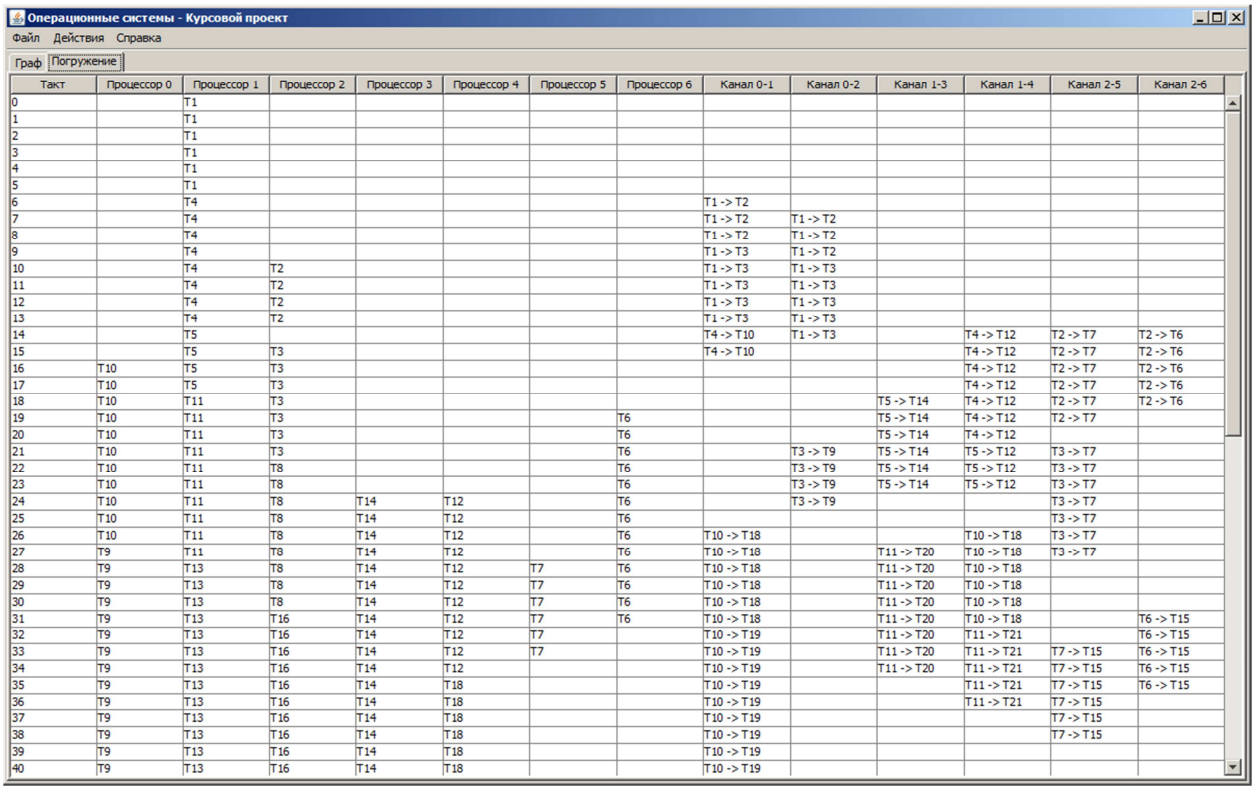


Рис. 4.2. Диалоговые окна для ввода весов новых вершины и ребра

Для запуска моделирования процесса планирования необходимо выбрать пункт «Погрузить граф...» (комбинация клавиш Ctrl-L) меню «Действия» и в диалоговом окне ввести количество процессоров в вычислительной системе. После этого в новой вкладке главного окна в виде диаграммы Ганта будут выведены результаты планирования (рис. 4.3).


Операционные системы - Курсовой проект

Файл Действия Справка

Граф | Погружение

Такт	Процессор 0	Процессор 1	Процессор 2	Процессор 3	Процессор 4	Процессор 5	Процессор 6	Канал 0-1	Канал 0-2	Канал 1-3	Канал 1-4	Канал 2-5	Канал 2-6
0		T1											
1		T1											
2		T1											
3		T1											
4		T1											
5		T1											
6		T4						T1 -> T2					
7		T4						T1 -> T2	T1 -> T2				
8		T4						T1 -> T2	T1 -> T2				
9		T4						T1 -> T3	T1 -> T2				
10		T4	T2					T1 -> T3	T1 -> T3				
11		T4	T2					T1 -> T3	T1 -> T3				
12		T4	T2					T1 -> T3	T1 -> T3				
13		T4	T2					T1 -> T3	T1 -> T3				
14		T5						T4 -> T10	T1 -> T3		T4 -> T12	T2 -> T7	T2 -> T6
15		T5	T3					T4 -> T10			T4 -> T12	T2 -> T7	T2 -> T6
16	T10	T5	T3								T4 -> T12	T2 -> T7	T2 -> T6
17	T10	T5	T3								T4 -> T12	T2 -> T7	T2 -> T6
18	T10	T11	T3							T5 -> T14	T4 -> T12	T2 -> T7	T2 -> T6
19	T10	T11	T3				T6			T5 -> T14	T4 -> T12	T2 -> T7	
20	T10	T11	T3				T6			T5 -> T14	T4 -> T12		
21	T10	T11	T3				T6		T3 -> T9	T5 -> T14	T5 -> T12	T3 -> T7	
22	T10	T11	T8				T6		T3 -> T9	T5 -> T14	T5 -> T12	T3 -> T7	
23	T10	T11	T8				T6		T3 -> T9	T5 -> T14	T5 -> T12	T3 -> T7	
24	T10	T11	T8	T14	T12		T6		T3 -> T9			T3 -> T7	
25	T10	T11	T8	T14	T12		T6					T3 -> T7	
26	T10	T11	T8	T14	T12		T6	T10 -> T18			T10 -> T18	T3 -> T7	
27	T9	T11	T8	T14	T12		T6	T10 -> T18		T11 -> T20	T10 -> T18	T3 -> T7	
28	T9	T13	T8	T14	T12	T7	T6	T10 -> T18		T11 -> T20	T10 -> T18		
29	T9	T13	T8	T14	T12	T7	T6	T10 -> T18		T11 -> T20	T10 -> T18		
30	T9	T13	T8	T14	T12	T7	T6	T10 -> T18		T11 -> T20	T10 -> T18		
31	T9	T13	T16	T14	T12	T7	T6	T10 -> T18		T11 -> T20	T10 -> T18		T6 -> T15
32	T9	T13	T16	T14	T12	T7		T10 -> T19		T11 -> T20	T11 -> T21		T6 -> T15
33	T9	T13	T16	T14	T12	T7		T10 -> T19		T11 -> T20	T11 -> T21	T7 -> T15	T6 -> T15
34	T9	T13	T16	T14	T12			T10 -> T19		T11 -> T20	T11 -> T21	T7 -> T15	T6 -> T15
35	T9	T13	T16	T14	T18			T10 -> T19			T11 -> T21	T7 -> T15	T6 -> T15
36	T9	T13	T16	T14	T18			T10 -> T19			T11 -> T21	T7 -> T15	
37	T9	T13	T16	T14	T18			T10 -> T19				T7 -> T15	
38	T9	T13	T16	T14	T18			T10 -> T19				T7 -> T15	
39	T9	T13	T16	T14	T18			T10 -> T19					
40	T9	T13	T16	T14	T18			T10 -> T19					

Рис. 4.3. Пример вывода результатов программы

5. Экспериментальные результаты работы программы

Для эксперимента использовался граф задач, изображенный на рисунке 5.1.

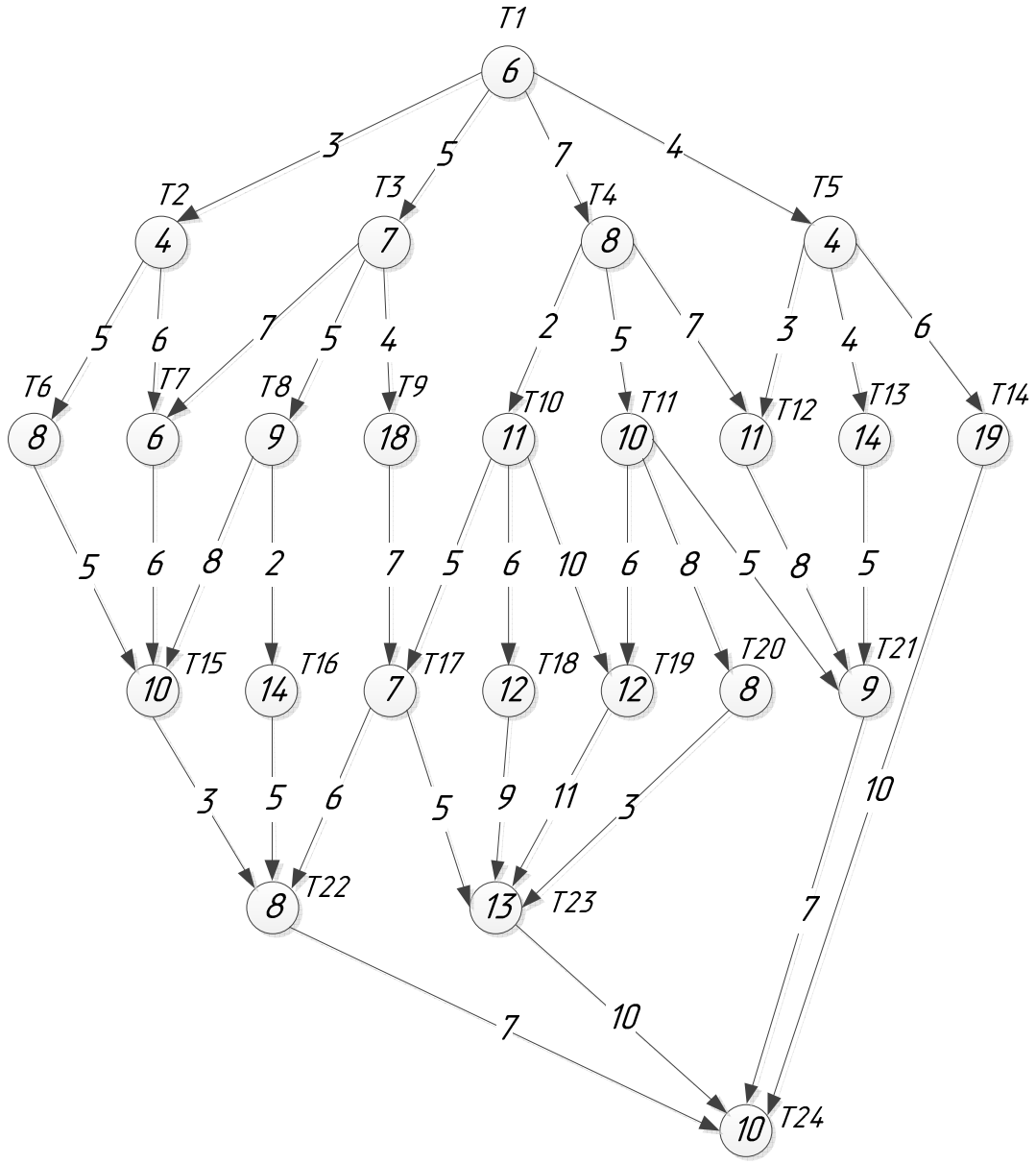


Рис. 5.1. Погружаемый граф задач (24 вершины)

Построим данный граф с помощью редактора графов созданной программы. Результат изображен на рисунке 5.2.

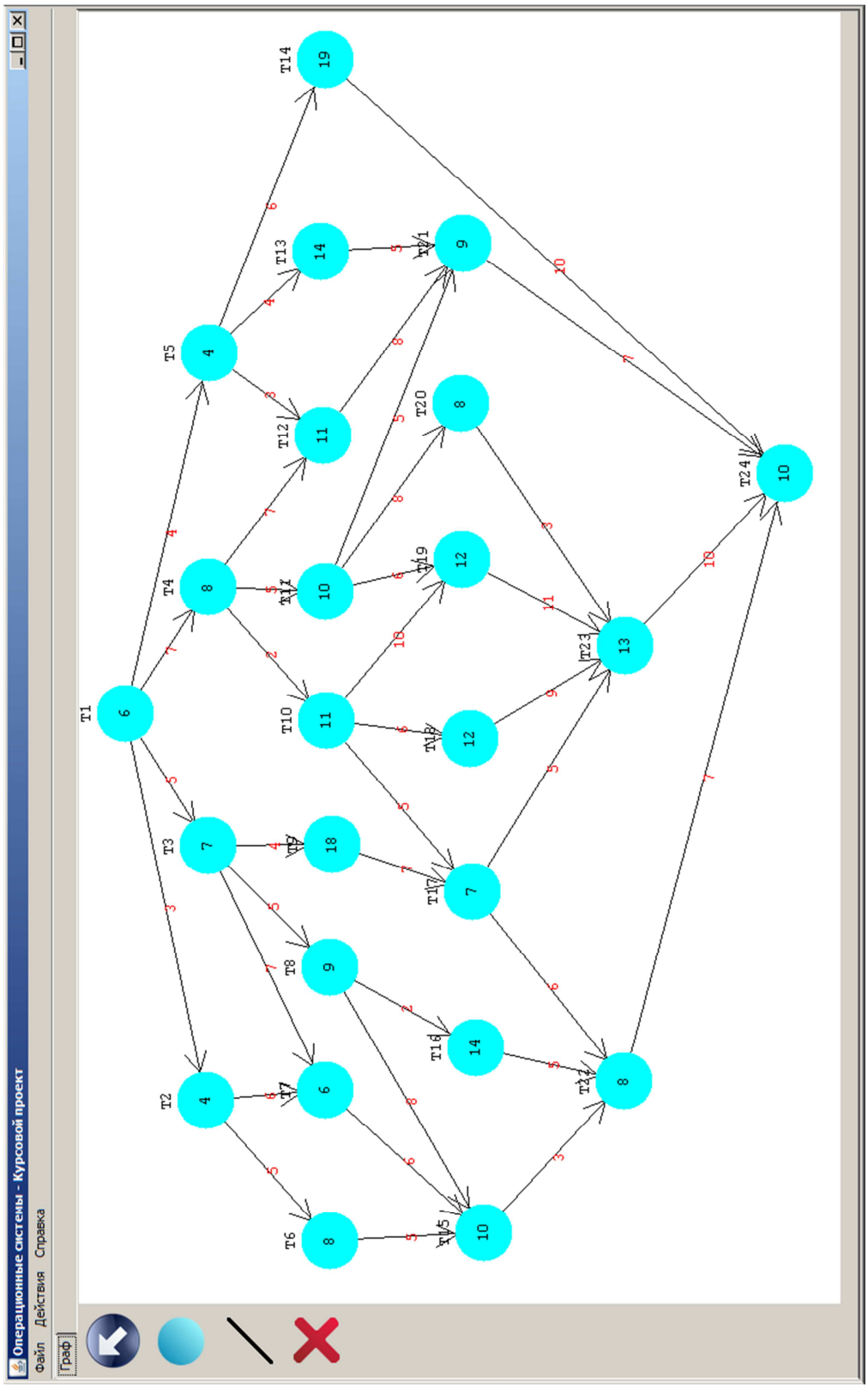
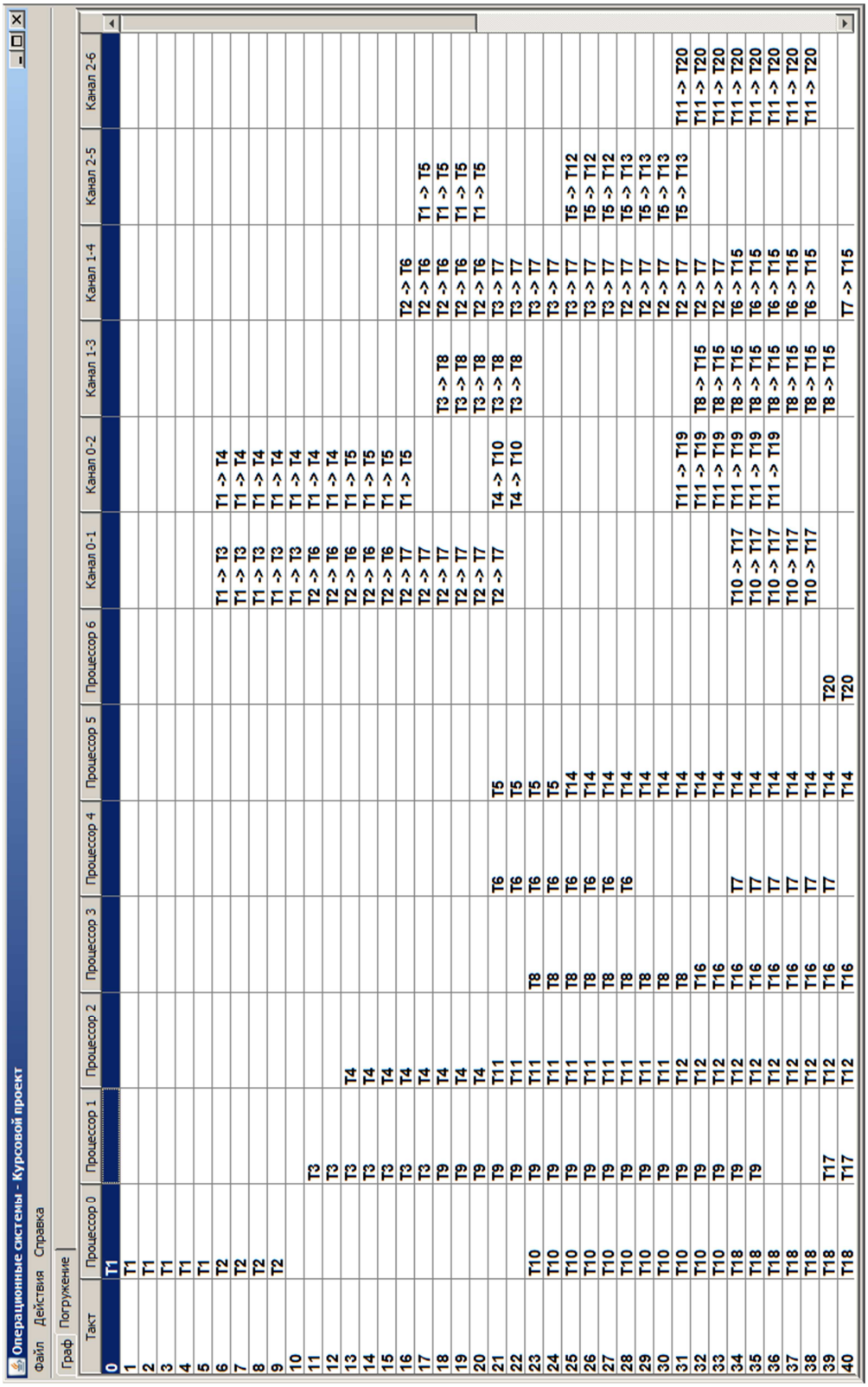


Рис. 5.2. Созданный в программе граф задач

Погрузим данный граф задач на вычислительную систему с топологией «бинарное дерево», состоящую из семи процессоров. Результаты представлены на рисунках 5.3, 5.4.



Такт	Процессор 0	Процессор 1	Процессор 2	Процессор 3	Процессор 4	Процессор 5	Процессор 6	Канал 0-1	Канал 0-2	Канал 1-3	Канал 1-4	Канал 2-5	Канал 2-6
0	T1												
1	T1												
2	T1												
3	T1												
4	T1												
5	T1												
6	T2							T1 -> T3	T1 -> T4				
7	T2							T1 -> T3	T1 -> T4				
8	T2							T1 -> T3	T1 -> T4				
9	T2							T1 -> T3	T1 -> T4				
10								T1 -> T3	T1 -> T4				
11		T3						T2 -> T6	T1 -> T4				
12		T3						T2 -> T6	T1 -> T4				
13		T3	T4					T2 -> T6	T1 -> T5				
14		T3	T4					T2 -> T6	T1 -> T5				
15		T3	T4					T2 -> T6	T1 -> T5				
16		T3	T4					T2 -> T7	T1 -> T5		T2 -> T6		
17		T3	T4					T2 -> T7			T2 -> T6	T1 -> T5	
18		T9	T4					T2 -> T7		T3 -> T8	T2 -> T6	T1 -> T5	
19		T9	T4					T2 -> T7		T3 -> T8	T2 -> T6	T1 -> T5	
20		T9	T4					T2 -> T7		T3 -> T8	T2 -> T6	T1 -> T5	
21		T9	T11		T6	T5		T2 -> T7	T4 -> T10	T3 -> T8	T3 -> T7		
22		T9	T11		T6	T5			T4 -> T10	T3 -> T8	T3 -> T7		
23	T10	T9	T11	T8	T6	T5					T3 -> T7		
24	T10	T9	T11	T8	T6	T5					T3 -> T7	T5 -> T12	
25	T10	T9	T11	T8	T6	T14					T3 -> T7	T5 -> T12	
26	T10	T9	T11	T8	T6	T14					T3 -> T7	T5 -> T12	
27	T10	T9	T11	T8	T6	T14					T3 -> T7	T5 -> T12	
28	T10	T9	T11	T8	T6	T14					T2 -> T7	T5 -> T13	
29	T10	T9	T11	T8		T14					T2 -> T7	T5 -> T13	
30	T10	T9	T11	T8		T14					T2 -> T7	T5 -> T13	
31	T10	T9	T12	T8		T14			T11 -> T19		T2 -> T7	T5 -> T13	T11 -> T20
32	T10	T9	T12	T16		T14			T11 -> T19	T8 -> T15	T2 -> T7		T11 -> T20
33	T10	T9	T12	T16		T14			T11 -> T19	T8 -> T15	T2 -> T7		T11 -> T20
34	T18	T9	T12	T16	T7	T14		T10 -> T17	T11 -> T19	T8 -> T15	T6 -> T15		T11 -> T20
35	T18	T9	T12	T16	T7	T14		T10 -> T17	T11 -> T19	T8 -> T15	T6 -> T15		T11 -> T20
36	T18		T12	T16	T7	T14		T10 -> T17	T11 -> T19	T8 -> T15	T6 -> T15		T11 -> T20
37	T18		T12	T16	T7	T14		T10 -> T17		T8 -> T15	T6 -> T15		T11 -> T20
38	T18		T12	T16	T7	T14		T10 -> T17		T8 -> T15	T6 -> T15		T11 -> T20
39	T18	T17	T12	T16	T7	T14	T20			T8 -> T15			
40	T18	T17	T12	T16		T14	T20				T7 -> T15		

Рис. 5.3. Результаты работы программы

Операционные системы - Курсовой проект

Файл Действия Справка

Граф | Погружение

Такт	Процессор 0	Процессор 1	Процессор 2	Процессор 3	Процессор 4	Процессор 5	Процессор 6	Канал 0-1	Канал 0-2	Канал 1-3	Канал 1-4	Канал 2-5	Канал 2-6
41	T18	T17	T12	T16		T14	T20				T7 -> T15		
42	T18	T17	T13	T16		T14	T20				T7 -> T15		
43	T18	T17	T13	T16		T14	T20				T7 -> T15		
44	T18	T17	T13	T16			T20				T7 -> T15	T14 -> T24	
45	T18	T17	T13	T16			T20				T7 -> T15	T14 -> T24	
46	T19	T15	T13				T20	T17 -> T23		T16 -> T22		T14 -> T24	
47	T19	T15	T13					T17 -> T23		T16 -> T22		T14 -> T24	T20 -> T23
48	T19	T15	T13					T17 -> T23		T16 -> T22		T14 -> T24	T20 -> T23
49	T19	T15	T13					T17 -> T23		T16 -> T22		T14 -> T24	T20 -> T23
50	T19	T15	T13					T17 -> T23	T20 -> T23	T16 -> T22		T14 -> T24	
51	T19	T15	T13						T20 -> T23			T14 -> T24	
52	T19	T15	T13						T20 -> T23			T14 -> T24	
53	T19	T15	T13									T14 -> T24	
54	T19	T15	T13						T14 -> T24				
55	T19	T15	T13						T14 -> T24				
56	T19	T22	T21						T14 -> T24				
57	T19	T22	T21						T14 -> T24				
58	T23	T22	T21						T14 -> T24				
59	T23	T22	T21						T14 -> T24				
60	T23	T22	T21						T14 -> T24				
61	T23	T22	T21						T14 -> T24				
62	T23	T22	T21						T14 -> T24				
63	T23	T22	T21						T14 -> T24				
64	T23		T21					T22 -> T24					
65	T23							T22 -> T24	T21 -> T24				
66	T23							T22 -> T24	T21 -> T24				
67	T23							T22 -> T24	T21 -> T24				
68	T23							T22 -> T24	T21 -> T24				
69	T23							T22 -> T24	T21 -> T24				
70	T23							T22 -> T24	T21 -> T24				
71									T21 -> T24				
72	T24												
73	T24												
74	T24												
75	T24												
76	T24												
77	T24												
78	T24												
79	T24												
80	T24												
81	T24												

Рис. 5.4. Результаты работы программы (продолжение)

6. Заключение

При выполнении курсового проекта мною были закреплены знания о принципах работы современных операционных систем, изучены различные методы решения задач пространственного планирования, оптимизации, рассмотрены основные этапы планирования, а также разработан алгоритм планирования для вычислительных систем с топологией «бинарное дерево».

Разработанный алгоритм представляет практический интерес при построении планировщиков многопроцессорных систем. В его разработке использован современный генетический подход. Качество работы алгоритма близко к идеальному (полного перебора), причем сложность алгоритма возрастает линейно со сложностью задачи. Данный алгоритм является универсальным и ориентирован для работы с распределенными и конфигурируемыми вычислительными системами. Дополнительным преимуществом данного метода является независимость от количества процессоров в вычислительной системе, так как количество процессоров в системе является входными данными для алгоритма.

Также была построена моделирующая программа, которая демонстрирует работу разработанного алгоритма на примере загрузки пользовательского графа задач на ресурсы системы. Листинг программы представлен в Приложении А.

Выполнение данного курсового проекта существенно улучшило мои навыки в прикладном программировании и помогло закрепить знания, полученные по курсу «Операционные системы».

Список использованной литературы

1. Конспект лекций по курсу «Операционные системы».
2. Русанова О.В., Нагорнюк В.В. Двухпроходной эвристический алгоритм планирования и отображения для систем с распределенной памятью // Вестник НТУУ "КПИ" "Информатика, управление и вычислительная техника". — 1998. — №31. — С. 238 —251.
3. Berman F. Experience with an automatic solution to the mapping problem // The Characteristics of Parallel Algorithms, MIT Press, Cambridge, MA, 1987. — P. 307 —334.
4. McFarland M., Parker A., Composano R. Tutorial on high-level synthesis // Proc. 25th Design Automation Conference. ACM and IEEE. — 1988. — P. 330 —336.
5. Князькова З.В., Симоненко В.П. Метод направленного поиска при статистическом планировании задач в распределенных вычислительных системах // Пробл. программирования. — 2002. — №1—2. — С. 247 —252.
6. Kramer O., Muhlenbein H. Mapping strategies in message based multiprocessor systems // Lecture Notes in Computer Science. — 1987. — 158. — P. 213 —225.
7. Shen H. Self-adjusting mapping: a heuristic mapping algorithm // Developing Transputer Applications, Amsterdam, IOS. — 1989. — P. 89 —98.
8. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / Г.К. Вороновкий, К.В. Махотило, С.Н. Петрашев, С.А. Сергеев. — ОСНОВА — 1997. — 112 с.
9. Исаев С.А. Генетические алгоритмы — эволюционные методы поиска.
10. Теория расписаний и вычислительные машины/ Под ред. Э.Г.Коффмана. М.: Наука, 1984. - 334с.

11. Гэри М., Джонсон Д. Вычислительные машины и трудно решаемые задачи. – М.: Мир, 1982. – 416с.
12. Шахбадзян К.В., Тушкина Т.А. Обзор методов составления расписаний для многопроцессорных систем// Записки научных семинаров ЛОМИ — Л.: Наука, 1975., т.54 — С.229–258.
13. Уоссермен Ф. Нейрокомпьютерная техника. Теория и практика.– М.: Мир, 1992.
14. Л.А. Растригин. Статистические методы поиска.– М.: Наука, 1968.

ПРИЛОЖЕНИЕ А

Класс Program предназначен для запуска программы.

```
import face.MainFrame;

import javax.swing.*;
import java.awt.*;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 26.12.10
 * Time: 16:33
 * To change this template use File | Settings | File Templates.
 */
public class Program {

    public static Rectangle getDefaultBounds() {
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        Dimension screenSize = toolkit.getScreenSize();
        Rectangle bounds = new Rectangle((int) screenSize.getWidth() / 12, (int)
screenSize.getHeight() / 12,
                (int) screenSize.getWidth() / 12 * 10, (int) screenSize.getHeight() /
12 * 10);
        return bounds;
    }

    public static void main(String[] args) {
        if (System.getProperty("os.name").toLowerCase().contains("windows")) {
            try {

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
            } catch (ClassNotFoundException e) {
            } catch (InstantiationException e) {
            } catch (IllegalAccessException e) {
            } catch (UnsupportedLookAndFeelException e) {
            }
        }
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                MainFrame frame = new MainFrame(Program.getDefaultBounds());
                frame.setVisible(true);
            }
        });
    }

}
```

Класс MainFrame реализует отрисовку главного окна программы и реализует основные действия с погружаемым графом.

```
package face;

import graph.GraphFileFilter;
import graph.GraphPanel;
import scheduler.Scheduler;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.io.*;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 26.12.10
 * Time: 16:43
 * To change this template use File | Settings | File Templates.
 */
```

```
public class MainFrame extends JFrame {

    private JMenuBar menuBar;
    private JTabbedPane tabbedPane;
    private JLabel statusLabel;
    private GraphPanel graphPanel;
    private JFileChooser chooser;
    private File openedFile;
    private JTable table;

    private NewAction newAction;
    private OpenAction openAction;
    private SaveAction saveAction;
    private SaveAsAction saveAsAction;
    private CloseAction closeAction;
    private ExitAction exitAction;
    private LoadAction loadAction;
    private AboutAction aboutAction;

    public MainFrame(Rectangle bounds) {
        super();
        setBounds(bounds);
        setMinimumSize(bounds.getSize());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Операционные системы - Курсовой проект");
        setLayout(new BorderLayout());
        newAction = new NewAction(this);
        openAction = new OpenAction(this);
        saveAction = new SaveAction(this);
        saveAsAction = new SaveAsAction(this);
        closeAction = new CloseAction(this);
        exitAction = new ExitAction(this);
        loadAction = new LoadAction(this);
        aboutAction = new AboutAction(this);
        chooser = new JFileChooser();
        chooser.setCurrentDirectory(new File("."));
        chooser.setMultiSelectionEnabled(false);
        menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("Файл");
        JMenu actionMenu = new JMenu("Действия");
        JMenu helpMenu = new JMenu("Справка");
        JMenuItem tempItem = new JMenuItem(newAction);
        tempItem.setText("Создать граф");
        tempItem.setMnemonic('N');
        tempItem.setAccelerator(KeyStroke.getKeyStroke('N', KeyEvent.CTRL_MASK));
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(openAction);
        tempItem.setText("Открыть граф...");
        tempItem.setMnemonic('O');
        tempItem.setAccelerator(KeyStroke.getKeyStroke('O', KeyEvent.CTRL_MASK));
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(saveAction);
        tempItem.setText("Сохранить граф");
        tempItem.setMnemonic('S');
        tempItem.setAccelerator(KeyStroke.getKeyStroke('S', KeyEvent.CTRL_MASK));
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(saveAsAction);
        tempItem.setText("Сохранить граф как...");
        tempItem.setMnemonic('S');
        tempItem.setAccelerator(KeyStroke.getKeyStroke('S', (KeyEvent.CTRL_MASK |
KeyEvent.SHIFT_MASK)));
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(closeAction);
        tempItem.setText("Закрыть граф");
        tempItem.setMnemonic('W');
        tempItem.setAccelerator(KeyStroke.getKeyStroke('W', KeyEvent.CTRL_MASK));
        fileMenu.add(tempItem);
        fileMenu.addSeparator();
        tempItem = new JMenuItem(exitAction);
        tempItem.setText("Выход");
        tempItem.setMnemonic('E');
```

```
        tempItem.setAccelerator(KeyStroke.getKeyStroke('E', KeyEvent.CTRL_MASK));
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(loadAction);
        tempItem.setText("Погрузить граф...");
        tempItem.setMnemonic('L');
        tempItem.setAccelerator(KeyStroke.getKeyStroke('L', KeyEvent.CTRL_MASK));
        actionMenu.add(tempItem);
        tempItem = new JMenuItem(aboutAction);
        tempItem.setText("О программе...");
        helpMenu.add(tempItem);
        menuBar.add(fileMenu);
        menuBar.add(actionMenu);
        menuBar.add(helpMenu);
        setJMenuBar(menuBar);
        tabbedPane = new JTabbedPane();
        graphPanel = new GraphPanel();
        addGraphTab();
        add(tabbedPane);
    }

    private void addGraphTab() {
        JPanel graphEditorPanel = new JPanel();
        graphEditorPanel.setLayout(new BorderLayout());
        JToolBar toolBar = new JToolBar(JToolBar.VERTICAL);
        toolBar.setFloatable(false);
        toolBar.setRollover(true);
        JButton tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                graphPanel.setAction((byte) 0);
                statusLabel.setText("Перемещение вершин");
            }
        });
        tempButton.setToolTipText("Перемещение вершин");
        tempButton.setIcon(new ImageIcon("img/no_action.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                graphPanel.setAction((byte) 1);
                statusLabel.setText("Добавление вершины");
            }
        });
        tempButton.setToolTipText("Добавление вершины");
        tempButton.setIcon(new ImageIcon("img/add_vertex.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                graphPanel.setAction((byte) 2);
                statusLabel.setText("Добавление ребра");
            }
        });
        tempButton.setToolTipText("Добавление ребра");
        tempButton.setIcon(new ImageIcon("img/add_connection.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                graphPanel.setAction((byte) 3);
                statusLabel.setText("Удаление вершины");
            }
        });
        tempButton.setToolTipText("Удаление вершины");
        tempButton.setIcon(new ImageIcon("img/delete_vertex.png"));
        graphEditorPanel.add(BorderLayout.WEST, toolBar);
        statusLabel = new JLabel(" ");
        graphEditorPanel.add(BorderLayout.SOUTH, statusLabel);
        graphEditorPanel.add(graphPanel);
        tabbedPane.addTab("Граф", graphEditorPanel);
    }

    private void addTableTab() {
        JPanel tablePanel = new JPanel();
        tablePanel.setLayout(new BorderLayout());
        tablePanel.add(new JScrollPane(table));
        tabbedPane.addTab("Погружение", tablePanel);
        tabbedPane.setSelectedIndex(tabbedPane.getTabCount() - 1);
```

```
    }

    private class NewAction extends AbstractAction {

        private MainFrame frame;

        public NewAction(MainFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            closeAction.actionPerformed(e);
            graphPanel = new GraphPanel();
            openedFile = null;
            addGraphTab();
        }

    }

    private class OpenAction extends AbstractAction {

        private MainFrame frame;

        public OpenAction(MainFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            chooser.resetChoosableFileFilters();
            chooser.addChoosableFileFilter(new GraphFileFilter());
            int result = chooser.showOpenDialog(frame);
            if (result == JFileChooser.APPROVE_OPTION) {
                try {
                    closeAction.actionPerformed(e);
                    ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(chooser.getSelectedFile()));
                    graphPanel = (GraphPanel) inputStream.readObject();
                    inputStream.close();
                    openedFile = new File(chooser.getSelectedFile().getName());
                    addGraphTab();
                } catch (IOException e1) {
                    JOptionPane.showMessageDialog(frame, "Произошла ошибка при
открытии файла.", "Ошибка",
                            JOptionPane.ERROR_MESSAGE);
                } catch (ClassNotFoundException e1) {
                    JOptionPane.showMessageDialog(frame, "Файл имеет не совместимый
формат.", "Ошибка",
                            JOptionPane.ERROR_MESSAGE);
                }
            }
        }

    }

    private class SaveAction extends AbstractAction {

        private MainFrame frame;

        public SaveAction(MainFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            if (openedFile == null) {
                saveAsAction.actionPerformed(e);
            }
            else {
                if (graphPanel.isModified()) {
                    try {
                        ObjectOutputStream outputStream = new ObjectOutputStream(new
FileOutputStream(openedFile));
```

```
                        graphPanel.setModified(false);
                        outputStream.writeObject(graphPanel);
                        outputStream.close();
                    } catch (IOException e1) {
                        JOptionPane.showMessageDialog(frame, "Произошла ошибка при
записи в файл.", "Ошибка",
                                JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        }

    }

    private class SaveAsAction extends AbstractAction {

        private MainFrame frame;

        public SaveAsAction(MainFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            chooser.resetChoosableFileFilters();
            chooser.addChoosableFileFilter(new GraphFileFilter());
            int result = chooser.showSaveDialog(frame);
            if (result == JFileChooser.APPROVE_OPTION) {
                if
(!chooser.getSelectedFile().getName().toLowerCase().endsWith(GraphFileFilter.GRAPH_EXT
ENSION)) {
                    chooser.setSelectedFile(new
File(chooser.getSelectedFile().getAbsolutePath() +
                            GraphFileFilter.GRAPH_EXTENSION));
                }
                try {
                    ObjectOutputStream outputStream = new ObjectOutputStream(new
FileOutputStream(chooser.getSelectedFile()));
                    graphPanel.setModified(false);
                    outputStream.writeObject(graphPanel);
                    outputStream.close();
                    openedFile = new File(chooser.getSelectedFile().getName());
                } catch (IOException e1) {
                    JOptionPane.showMessageDialog(frame, "Произошла ошибка при
создании файла.", "Ошибка",
                            JOptionPane.ERROR_MESSAGE);
                }
            }
        }

    }

    private class CloseAction extends AbstractAction {

        private MainFrame frame;

        public CloseAction(MainFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            if ((graphPanel != null) && (graphPanel.isModified())) {
                int result = JOptionPane.showConfirmDialog(frame,
                        "Граф содержит несохраненные изменения. Желаете сохранить их
перед закрытием?",
                        "Подтверждение", JOptionPane.YES_NO_OPTION);
                if (result == JOptionPane.YES_OPTION) {
                    if (openedFile != null) {
                        saveAction.actionPerformed(e);
                    }
                    else {
                        saveAsAction.actionPerformed(e);
```

```
                    }
                }
            }
            openedFile = null;
            graphPanel = null;
            while (tabbedPane.getTabCount() > 0) {
                tabbedPane.remove(0);
            }
        }

    }

    private class ExitAction extends AbstractAction {

        private MainFrame frame;

        public ExitAction(MainFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            closeAction.actionPerformed(e);
            System.exit(0);
        }

    }

    private class LoadAction extends AbstractAction {

        private MainFrame frame;

        public LoadAction(MainFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            if (graphPanel == null) {
                 JOptionPane.showMessageDialog(frame, "Сначала необходимо создать граф
для погружения.", "Ошибка",
                            JOptionPane.ERROR_MESSAGE);
            } else {
                String processorsCountString = JOptionPane.showInputDialog(frame,
                        "Введите количество процессоров в вычислительной системе:");
                int processorsCount = Integer.valueOf(processorsCountString);
                Scheduler scheduler = new Scheduler(processorsCount);
                if (tabbedPane.getTabCount() > 1) {
                    tabbedPane.remove(tabbedPane.getTabCount() - 1);
                    table = null;
                }
                table = new
JTable(scheduler.loadGraph(graphPanel.getConnectionMatrix(),
graphPanel.getVertexesWeightVector(),
                        graphPanel.getVertexesNameVector()));
                addTableTab();
            }
        }

    }

    private class AboutAction extends AbstractAction {

        private MainFrame frame;

        public AboutAction(MainFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(frame,
                    "Операционные системы - Курсовой проект\nCopyright (c) 2011
Захожий Игорь",
```

```
                    "О программе", JOptionPane.INFORMATION_MESSAGE);
        }

    }

}
```

Классы ConnectionAlreadyExistException и ConnectionDoesntExistException предназначены для обработки ошибок при пристроении графа.

```
package graph;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 26.12.10
 * Time: 19:15
 * To change this template use File | Settings | File Templates.
 */
class ConnectionAlreadyExistException extends Exception {

    private static String TEXT = "Связь уже существует.";

    @Override
    public String getMessage() {
        return TEXT;
    }

}
```

```
package graph;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 26.12.10
 * Time: 19:18
 * To change this template use File | Settings | File Templates.
 */
class ConnectionDoesntExistException extends Exception {

    private static String TEXT = "Связь не существует.";

    @Override
    public String getMessage() {
        return TEXT;
    }


}
```

Класс GraphFileFilter предназначен для выбора файлов необходимого формата при открытии созданного ранее графа.

```
package graph;

import javax.swing.filechooser.FileFilter;
import java.io.File;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 27.12.10
 * Time: 10:19
 * To change this template use File | Settings | File Templates.
 */
public class GraphFileFilter extends FileFilter {
```

```
    public static String GRAPH_EXTENSION = ".graph";

    private static String GRAPH_DESCRIPTION = "Graph File";

    public boolean accept(File pathname) {
        return (pathname.getName().toLowerCase().endsWith(GRAPH_EXTENSION) ||
pathname.isDirectory());
    }

    public String getDescription() {
        return GRAPH_DESCRIPTION;
    }

}
```

Класс GraphPanel представляет панель для отображения, построения и редактирования погружаемого графа. Классы Vertex и Line – вспомогательные классы.

```
package graph;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Arrays;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 26.12.10
 * Time: 16:50
 * To change this template use File | Settings | File Templates.
 */
public class GraphPanel extends JPanel implements Serializable {

    private static Color PANEL_COLOR = Color.WHITE;

    private ArrayList<Vertex> vertexes;
    private ArrayList<Line> lines;

    private boolean modified;

    private byte action;
    // 0 - Moving vertexes
    // 1 - Adding vertex
    // 2 - Adding connection
    // 3 - Deleting vertex

    private Vertex draggedVertex;
    private Point tempPoint;
    private Line tempLine;
    private Vertex tempVertex;

    public GraphPanel() {
        super();
        addMouseListener(new GraphMouseListener(this));
        addMouseMotionListener(new GraphMouseMotionListener(this));
        setBackground(PANEL_COLOR);
        lines = new ArrayList<Line>();
        vertexes = new ArrayList<Vertex>();
        modified = false;
        action = 0;
        draggedVertex = null;
        tempPoint = null;
```

```
        tempLine = null;
        tempVertex = null;
    }

    public void setAction(byte action) {
        this.action = action;
    }

    public boolean isModified() {
        return modified;
    }

    public void setModified(boolean modified) {
        this.modified = modified;
    }

    public String[] getVertexesNameVector() {
        String[] vector = new String[vertexes.size()];
        for (int i = 0; i < vertexes.size(); i++) {
            vector[i] = new String(vertexes.get(i).getName());
        }
        return vector;
    }

    public int[] getVertexesWeightVector() {
        int[] vector = new int[vertexes.size()];
        for (int i = 0; i < vertexes.size(); i++) {
            vector[i] = vertexes.get(i).getWeight();
        }
        return vector;
    }

    public int[][] getConnectionMatrix() {
        int[][] matrix = new int[vertexes.size()][];
        for (int i = 0; i < matrix.length; i++) {
            matrix[i] = new int[vertexes.size()];
            for (int j = 0; j < matrix[i].length; j++) {
                matrix[i][j] = 0;
            }
        }
        for (int i = 0; i < vertexes.size(); i++) {
            ArrayList<Vertex> nextVertexes = vertexes.get(i).getNextVertexes();
            for (int j = 0; j < nextVertexes.size(); j++) {
                int k = 0;
                boolean found = false;
                while ((k < vertexes.size()) && (!found)) {
                    if (vertexes.get(k) == nextVertexes.get(j)) {
                        found = true;
                    }
                    k++;
                }
                if (found) {
                    k--;
                    boolean lineFound = false;
                    int lineWeight = 0;
                    int l = 0;
                    while ((!lineFound) && (l < lines.size())) {
                        if ((lines.get(l).getP1().getX() ==
vertexes.get(i).getCenter().getX()) &&
                                (lines.get(l).getP1().getY() ==
vertexes.get(i).getCenter().getY()) &&
                                (lines.get(l).getP2().getX() ==
vertexes.get(k).getCenter().getX()) &&
                                (lines.get(l).getP2().getY() ==
vertexes.get(k).getCenter().getY())) {
                            lineFound = true;
                            lineWeight = lines.get(l).getWeight();
                        }
                        l++;
                    }
                    if (lineFound) {
```

```
                    matrix[i][k] = lineWeight;
                }
            }
        }
    }
    return matrix;
}

public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    if (!lines.isEmpty()) {
        for (Line l : lines) {
            l.draw(g2);
        }
    }
    if (tempLine != null) {
        tempLine.draw(g2);
    }
    if (!vertexes.isEmpty()) {
        for (Vertex v : vertexes) {
            v.draw(g2);
        }
    }
}

private class GraphMouseListener extends MouseAdapter implements Serializable {

    private GraphPanel panel;

    private GraphMouseListener(GraphPanel panel) {
        this.panel = panel;
    }

    @Override
    public void mousePressed(MouseEvent e) {
        switch (action) {
            case 0: {
                int i = 0;
                while ((i < vertexes.size()) && (draggedVertex == null)) {
                    if (vertexes.get(i).contains(e.getPoint())) {
                        draggedVertex = vertexes.get(i);
                    }
                    i++;
                }
                break;
            }
            case 2: {
                int i = 0;
                while ((i < vertexes.size()) && (tempPoint == null)) {
                    if (vertexes.get(i).contains(e.getPoint())) {
                        tempPoint = vertexes.get(i).getCenter();
                        tempVertex = vertexes.get(i);
                    }
                    i++;
                }
                break;
            }
        }
    }

    @Override
    public void mouseReleased(MouseEvent e) {
        switch (action) {
            case 0: {
                draggedVertex = null;
                break;
            }
            case 2: {
                int i = 0;
                boolean found = false;
```

```
                    while ((i < vertexes.size()) && (!found)) {
                        if (vertexes.get(i).contains(e.getPoint())) {
                            found = true;
                        }
                        i++;
                    }
                    if (found) {
                        i--;
                        String weightString = JOptionPane.showInputDialog(panel,
"Введите вес ребра:");
                        int weight = Integer.valueOf(weightString);
                        if (weight > 0) {
                            lines.add(new Line(tempPoint, vertexes.get(i).getCenter(),
weight));
                            try {
                                tempVertex.addNextVertex(vertexes.get(i));
                                modified = true;
                            } catch (ConnectionAlreadyExistException e1) {
                                JOptionPane.showMessageDialog(panel, e1.getMessage(),
"Ошибка",
                                        JOptionPane.ERROR_MESSAGE);
                            }
                        }
                        else {
                            JOptionPane.showMessageDialog(panel, "Некорректный вес
ребра!", "Ошибка",
                                    JOptionPane.ERROR_MESSAGE);
                        }
                    }
                    tempPoint = null;
                    tempLine = null;
                    tempVertex = null;
                    repaint();
                    break;
                }
            }
        }

        @Override
        public void mouseClicked(MouseEvent e) {
            if (e.getButton() == MouseEvent.BUTTON1) {
                switch (action) {
                    case 1: {
                        String name = JOptionPane.showInputDialog(panel, "Введите имя
вершины:");
                        String weightString = JOptionPane.showInputDialog(panel,
"Введите вес вершины:");
                        int weight = Integer.valueOf(weightString);
                        if (weight > 0) {
                            vertexes.add(new Vertex(name, weight, e.getPoint()));
                            modified = true;
                            repaint();
                        }
                        else {
                            JOptionPane.showMessageDialog(panel, "Некорректный вес
вершины!", "Ошибка",
                                    JOptionPane.ERROR_MESSAGE);
                        }
                        break;
                    }
                    case 3: {
                        int i = 0;
                        boolean found = false;
                        while ((i < vertexes.size()) && (!found)) {
                            if (vertexes.get(i).contains(e.getPoint())) {
                                found = true;
                            }
                            i++;
                        }
                        if (found) {
```

```
                            int result = JOptionPane.showConfirmDialog(panel, "Вы
действительно хотите удалить эту вершину?",
                                    "Подтверждение удаления",
JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
                            if (result == JOptionPane.YES_OPTION) {
                                i--;
                                for (Vertex v : vertexes) {
                                    try {
                                        v.removeNextVertex(vertexes.get(i));
                                    } catch (ConnectionDoesntExistException e1) {}
                                }
                                for (int j = lines.size() - 1; j >= 0; j--) {
                                    if (((lines.get(j).getP1().getX() ==
vertexes.get(i).getCenter().getX()) &&
                                            (lines.get(j).getP1().getY() ==
vertexes.get(i).getCenter().getY())) ||
                                            ((lines.get(j).getP2().getX() ==
vertexes.get(i).getCenter().getX()) &&
                                            (lines.get(j).getP2().getY() ==
vertexes.get(i).getCenter().getY()))) {
                                        lines.remove(j);
                                    }
                                }
                                vertexes.remove(i);
                                modified = true;
                                repaint();
                            }
                        }
                        break;
                    }
                }
            }
            else {
                if (e.getButton() == MouseEvent.BUTTON3) {
                    int i = 0;
                    boolean found = false;
                    while ((i < vertexes.size()) && (!found)) {
                        if (vertexes.get(i).contains(e.getPoint())) {
                            found = true;
                        }
                        i++;
                    }
                    if (found) {
                        i--;
                        JPopupMenu popupMenu = new JPopupMenu();
                        JMenu deleteConnectionMenu = new JMenu("Удалить ребро...");
                        for (Vertex v : vertexes.get(i).getNextVertexes()) {
                            JMenuItem menuItem = new JMenuItem(new
DeleteConnectionAction(vertexes.get(i), v));
                            StringBuilder builder = new StringBuilder();
                            builder.append(vertexes.get(i).getName());
                            builder.append(" -> ");
                            builder.append(v.getName());
                            menuItem.setText(builder.toString());
                            deleteConnectionMenu.add(menuItem);
                        }
                        popupMenu.add(deleteConnectionMenu);
                        popupMenu.show(panel, e.getX(), e.getY());
                    }
                }
            }
        }

        private class DeleteConnectionAction extends AbstractAction implements
Serializable {

            private Vertex parentVertex;
            private Vertex childVertex;

            public DeleteConnectionAction(Vertex parentVertex, Vertex childVertex) {
                this.parentVertex = parentVertex;
```

```
                this.childVertex = childVertex;
            }

            public void actionPerformed(ActionEvent e) {
                try {
                    parentVertex.removeNextVertex(childVertex);
                } catch (ConnectionDoesntExistException e1) {}
                int i = 0;
                boolean found = false;
                while ((!found) && (i < lines.size())) {
                    if ((lines.get(i).getP1().getX() ==
parentVertex.getCenter().getX()) &&
                            (lines.get(i).getP1().getY() ==
parentVertex.getCenter().getY()) &&
                            (lines.get(i).getP2().getX() ==
childVertex.getCenter().getX()) &&
                            (lines.get(i).getP2().getY() ==
childVertex.getCenter().getY())) {
                        found = true;
                    }
                    i++;
                }
                if (found) {
                    lines.remove(--i);
                    modified = true;
                }
                repaint();
            }

        }

    }

    private class GraphMouseMotionListener extends MouseMotionAdapter implements
Serializable {

        private GraphPanel panel;

        private GraphMouseMotionListener(GraphPanel panel) {
            this.panel = panel;
        }

        @Override
        public void mouseDragged(MouseEvent e) {
            if (draggedVertex != null) {
                ArrayList<Line> inLines = new ArrayList<Line>();
                ArrayList<Line> outLines = new ArrayList<Line>();
                for (Line l : lines) {
                    if ((l.getP1().getX() == draggedVertex.getCenter().getX()) &&
                            (l.getP1().getY() == draggedVertex.getCenter().getY())) {
                        outLines.add(l);
                    }
                    else {
                        if ((l.getP2().getX() == draggedVertex.getCenter().getX()) &&
                                (l.getP2().getY() ==
draggedVertex.getCenter().getY())) {
                            inLines.add(l);
                        }
                    }
                }
                draggedVertex.setLeftUpperCorner(e.getPoint());
                for (Line l : inLines) {
                    l.setP2(draggedVertex.getCenter());
                }
                for (Line l : outLines) {
                    l.setP1(draggedVertex.getCenter());
                }
                modified = true;
                repaint();
            }
            if (tempPoint != null) {
```

```
                tempLine = new Line(tempPoint, e.getPoint(), -1);
                repaint();
            }
        }

    }

}

package graph;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.io.Serializable;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 26.12.10
 * Time: 18:52
 * To change this template use File | Settings | File Templates.
 */
class Line implements Serializable {

    private static Color LINE_COLOR = Color.BLACK;
    private static Color TEXT_COLOR = Color.RED;
    private static Font TEXT_FONT = new Font("Monospaced", Font.PLAIN, 14);
    private static int ARROW_LENGTH = 20;
    private static double ARROW_ANGLE = 0.5;

    private Point2D p1;
    private Point2D p2;
    private int weight;

    public Line(Point2D p1, Point2D p2, int weight) {
        this.p1 = p1;
        this.p2 = p2;
        this.weight = weight;
    }

    public Point2D getP1() {
        return p1;
    }

    public void setP1(Point2D p1) {
        this.p1 = p1;
    }

    public Point2D getP2() {
        return p2;
    }

    public void setP2(Point2D p2) {
        this.p2 = p2;
    }

    public int getWeight() {
        return weight;
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public void draw(Graphics2D g2) {
        g2.setColor(LINE_COLOR);
        g2.drawLine((int) p1.getX(), (int) p1.getY(), (int) p2.getX(), (int)
p2.getY());
```

```
        int length = (int) Math.sqrt(Math.pow(p2.getX() - p1.getX(), 2) +
Math.pow(p2.getY() - p1.getY(), 2));
        int lambda = (int) (length / (Vertex.getVERTEX_DIAMETER() / 2));
        int arrowX = (int) ((p1.getX() + lambda * p2.getX()) / (1 + lambda));
        int arrowY = (int) ((p1.getY() + lambda * p2.getY()) / (1 + lambda));
        double temp = Math.atan2(p1.getX() - arrowX, p1.getY() - arrowY);
        g2.drawLine(arrowX, arrowY,
                (int) (arrowX + ARROW_LENGTH * Math.sin(temp + ARROW_ANGLE)),
                (int) (arrowY + ARROW_LENGTH * Math.cos(temp + ARROW_ANGLE)));
        g2.drawLine(arrowX, arrowY,
                (int) (arrowX + ARROW_LENGTH * Math.sin(temp - ARROW_ANGLE)),
                (int) (arrowY + ARROW_LENGTH * Math.cos(temp - ARROW_ANGLE)));
        if (weight != -1) {
            g2.setColor(TEXT_COLOR);
            g2.setFont(TEXT_FONT);
            FontRenderContext context = g2.getFontRenderContext();
            String weightString = String.valueOf(weight);
            int centerX = (int) ((p1.getX() + p2.getX()) / 2);
            int centerY = (int) ((p1.getY() + p2.getY()) / 2);
            Rectangle2D bounds = TEXT_FONT.getStringBounds(weightString, context);
            int x = centerX - (int) bounds.getWidth() / 2;
            int y = centerY + (int) bounds.getHeight() / 2;
            g2.drawString(weightString, x, y);
        }
    }

}

package graph;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.io.Serializable;
import java.util.ArrayList;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 26.12.10
 * Time: 18:25
 * To change this template use File | Settings | File Templates.
 */
class Vertex implements Serializable {

    private static Color VERTEX_COLOR = Color.CYAN;
    private static Color TEXT_COLOR = Color.BLACK;
    private static int VERTEX_DIAMETER = 50;
    private static Font TEXT_FONT = new Font("Monospaced", Font.PLAIN, 14);

    private String name;
    private int weight;
    private Point2D leftUpperCorner;

    private ArrayList<Vertex> nextVertexes;

    public Vertex(String name, int weight, Point2D leftUpperCorner) {
        this.name = name;
        this.weight = weight;
        this.leftUpperCorner = leftUpperCorner;
        nextVertexes = new ArrayList<Vertex>();
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public int getWeight() {
        return weight;
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public Point2D getLeftUpperCorner() {
        return leftUpperCorner;
    }

    public void setLeftUpperCorner(Point2D leftUpperCorner) {
        this.leftUpperCorner = leftUpperCorner;
    }

    public ArrayList<Vertex> getNextVertexes() {
        return nextVertexes;
    }

    public static int getVERTEX_DIAMETER() {
        return VERTEX_DIAMETER;
    }

    public void addNextVertex(Vertex v) throws ConnectionAlreadyExistException {
        if (nextVertexes.contains(v)) {
            throw new ConnectionAlreadyExistException();
        }
        nextVertexes.add(v);
    }

    public void removeNextVertex(Vertex v) throws ConnectionDoesntExistException {
        if (!nextVertexes.contains(v)) {
            throw new ConnectionDoesntExistException();
        }
        nextVertexes.remove(v);
    }

    public boolean contains(Point p) {
        if ((p.getX() >= leftUpperCorner.getX()) && (p.getX() <=
leftUpperCorner.getX() + VERTEX_DIAMETER) &&
                (p.getY() >= leftUpperCorner.getY()) && (p.getY() <=
leftUpperCorner.getY() + VERTEX_DIAMETER)) {
            return true;
        }
        else {
            return false;
        }
    }

    public Point getCenter() {
        return new Point((int) (leftUpperCorner.getX() + (VERTEX_DIAMETER / 2)),
                (int) (leftUpperCorner.getY() + (VERTEX_DIAMETER / 2)));
    }

    public void draw(Graphics2D g2) {
        g2.setColor(VERTEX_COLOR);
        g2.fillOval((int) leftUpperCorner.getX(), (int) leftUpperCorner.getY(),
VERTEX_DIAMETER, VERTEX_DIAMETER);
        g2.setColor(TEXT_COLOR);
        g2.setFont(TEXT_FONT);
        FontRenderContext context = g2.getFontRenderContext();
        String weightString = String.valueOf(weight);
        Rectangle2D bounds = TEXT_FONT.getStringBounds(weightString, context);
        int x = ((int) leftUpperCorner.getX() + VERTEX_DIAMETER / 2 - (int)
bounds.getWidth() / 2);
        int y = ((int) leftUpperCorner.getY() + VERTEX_DIAMETER / 2 + (int)
bounds.getHeight() / 4);
        g2.drawString(weightString, x ,y);
        bounds = TEXT_FONT.getStringBounds(name, context);
```

```
        x = ((int) leftUpperCorner.getX() + VERTEX_DIAMETER / 2 - (int)
bounds.getWidth() / 2);
        y = (int) leftUpperCorner.getY() - 5;
        g2.drawString(name, x, y);
    }

}
```

Класс GanttTableModel представляет собой модель таблицы для вывода результатов планирования.

```
package scheduler;

import javax.swing.table.AbstractTableModel;
import java.util.ArrayList;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 27.12.10
 * Time: 17:38
 * To change this template use File | Settings | File Templates.
 */
public class GanttTableModel extends AbstractTableModel {

    private int columnCount;
    private String[] columnNames;
    private ArrayList<String[]> table;

    public GanttTableModel(int columnCount) {
        this.columnCount = columnCount;
        table = new ArrayList<String[]>();
    }

    public void addColumnNamesRow(String[] row) {
        columnNames = row;
    }

    public void addRow(String[] row) {
        table.add(row);
    }

    public void setValueAt(String value, int row, int column) {
        while (row >= table.size()) {
             table.add(new String[columnCount]);
        }
        table.get(row)[column] = value;
    }

    public int getRowCount() {
        return table.size();
    }

    public int getColumnCount() {
        return columnCount;
    }

    @Override
    public String getColumnName(int column) {
        return columnNames[column];
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        return table.get(rowIndex)[columnIndex];
    }

}
```

Класс Scheduler содержит все методы, реализующие планирование.

```
package scheduler;

import java.util.ArrayList;

/**
 * Created by IntelliJ IDEA.
 * User: Zakhozhyy Ihor
 * Date: 27.12.10
 * Time: 17:25
 * To change this template use File | Settings | File Templates.
 */
public class Scheduler {

    private static int STEP_COUNT = 5;
    private static double CROSS_PROBABILITY = 0.5;
    private static double MUTATION_PROBABILITY = 0.05;
    private static int POPULATION_STEP = 10;
    private static int BEST_GENOTYPE_NOT_CHANGED_MAX_COUNT = 5;

    private int processorCount;
    private int channelCount;
    private int populationSize;

    public Scheduler(int processorCount) {
        this.processorCount = processorCount;
        channelCount = processorCount - 1;
        populationSize = POPULATION_STEP * processorCount;
    }

    private ArrayList<Integer> getReadyTasks(int[][] connectionMatrix) {
        ArrayList<Integer> readyTasks = new ArrayList<Integer>();
        for (int i = 0; i < connectionMatrix[0].length; i++) {
            boolean ready = true;
            int j = 0;
            while ((ready) && (j < connectionMatrix.length)) {
                if (connectionMatrix[j][i] > 0) {
                    ready = false;
                }
                j++;
            }
            if (ready) {
                readyTasks.add(i);
            }
        }
        return readyTasks;
    }

    private boolean isAllTasksDone(int[] vertexWeight) {
        boolean result = true;
        int i = 0;
        while ((i < vertexWeight.length) && (result)) {
            if (vertexWeight[i] > 0) {
                result = false;
            }
            i++;
        }
        return result;
    }

    private ArrayList<Integer> getWay(int from, int to) {
        ArrayList<Integer> way = new ArrayList<Integer>();
        ArrayList<ArrayList<Integer>> allWays = new ArrayList<ArrayList<Integer>>();
        int waysCount = (processorCount + 1) / 2;
        int waysLength = (int) (Math.log(processorCount + 1) / Math.log(2));
        for (int i = 0; i < waysCount; i++) {
            allWays.add(new ArrayList<Integer>());
            for (int j = 0; j < waysLength; j++) {
                allWays.get(i).add(0);
            }
            allWays.get(i).set(0, 0);
```

```
        }
        int processorNumber = 1;
        int counter = 0;
        int counterEdge = waysCount / 2;
        for (int i = 0; i < (waysLength - 1); i++) {
            for (int j = 0; j < waysCount; j++) {
                allWays.get(j).set(i + 1, processorNumber);
                counter++;
                if (counter >= counterEdge) {
                    processorNumber++;
                    counter = 0;
                }
            }
            counterEdge /= 2;
        }
        int foundFrom = -1;
        int foundTo = -1;
        int fromIndex = -1;
        int toIndex = -1;
        int i = 0;
        while ((i < allWays.size()) && ((foundFrom < 0) || (foundTo < 0))) {
            for (int j = 0; j < allWays.get(i).size(); j++) {
                if (allWays.get(i).get(j) == from) {
                    foundFrom = i;
                    fromIndex = j;
                }
                if (allWays.get(i).get(j) == to) {
                    foundTo = i;
                    toIndex = j;
                }
            }
            i++;
        }
        if (foundFrom == foundTo) {
            if (fromIndex < toIndex) {
                for (int j = fromIndex; j <= toIndex; j++) {
                    way.add(allWays.get(foundFrom).get(j));
                }
            }
            else {
                for (int j = fromIndex; j >= toIndex; j--) {
                    way.add(allWays.get(foundFrom).get(j));
                }
            }
        }
        else {
            for (int j = fromIndex; j > 0; j--) {
                way.add(allWays.get(foundFrom).get(j));
            }
            for (int j = 0; j <= toIndex; j++) {
                way.add(allWays.get(foundTo).get(j));
            }
        }
        return way;
    }

    private int getExecutionTime(ArrayList<Integer> processorNumbers, int[][]
originalConnectionMatrix,
                                 int[] originalVertexWeight) {
        int[][] connectionMatrix = new int[originalConnectionMatrix.length][];
        for (int i = 0; i < connectionMatrix.length; i++) {
            connectionMatrix[i] = new int[originalConnectionMatrix[i].length];
            for (int j = 0; j < connectionMatrix[i].length; j++) {
                connectionMatrix[i][j] = originalConnectionMatrix[i][j];
            }
        }
        int[] vertexWeight = new int[originalVertexWeight.length];
        for (int i = 0; i < vertexWeight.length; i++) {
            vertexWeight[i] = originalVertexWeight[i];
        }
        int time = 0;
```

```
        boolean[] processorsStatus = new boolean[processorCount];
        for (boolean e : processorsStatus) {
            e = false;
        }
        int[] processorsTask = new int[processorCount];
        for (int e : processorsTask) {
            e = -1;
        }
        int[] channelProcessorFrom = new int[channelCount];
        int[] channelProcessorTo = new int[channelCount];
        int fromProcessor = 0;
        int tempCounter = 0;
        for (int j = 0; j < channelCount; j++) {
            channelProcessorFrom[j] = fromProcessor;
            channelProcessorTo[j] = j + 1;
            tempCounter++;
            if (tempCounter > 1) {
                tempCounter = 0;
                fromProcessor++;
            }
        }
        boolean[] channelsStatus = new boolean[channelCount];
        for (boolean e : channelsStatus) {
            e = false;
        }
        int[] channelFrom = new int[channelCount];
        for (int e : channelFrom) {
            e = -1;
        }
        int[] channelTo = new int[channelCount];
        for (int e : channelTo) {
            e = -1;
        }
        ArrayList<ArrayList<Boolean>> startNextTact = new
ArrayList<ArrayList<Boolean>>();
        boolean[] isAlreadySend = new boolean[channelCount];
        int[] channelTaskWeight = new int[channelCount];
        int[] channelTaskWeightBackup = new int[channelCount];
        ArrayList<ArrayList<Integer>> channelTaskWay = new
ArrayList<ArrayList<Integer>>();
        ArrayList<ArrayList<Integer>> channelFromQueue = new
ArrayList<ArrayList<Integer>>();
        ArrayList<ArrayList<Integer>> channelToQueue = new
ArrayList<ArrayList<Integer>>();
        ArrayList<ArrayList<ArrayList<Integer>>> channelWayQueue = new
ArrayList<ArrayList<ArrayList<Integer>>>();
        ArrayList<ArrayList<Integer>> channelWeightQueue = new
ArrayList<ArrayList<Integer>>();
        for (int i = 0; i < channelCount; i++) {
            startNextTact.add(new ArrayList<Boolean>());
            isAlreadySend[i] = false;
            channelTaskWeight[i] = -1;
            channelTaskWay.add(new ArrayList<Integer>());
            channelFromQueue.add(new ArrayList<Integer>());
            channelToQueue.add(new ArrayList<Integer>());
            channelWayQueue.add(new ArrayList<ArrayList<Integer>>());
            channelWeightQueue.add(new ArrayList<Integer>());
        }
        while (!isAllTasksDone(vertexWeight)) {
            for (int i = 0; i < channelCount; i++) {
                for (int j = 0; j < startNextTact.get(i).size(); j++) {
                    startNextTact.get(i).set(j, false);
                }
            }
            ArrayList<Integer> readyTasks = getReadyTasks(connectionMatrix);
            for (int i = 0; i < readyTasks.size(); i++) {
                if (vertexWeight[readyTasks.get(i)] > 0) {
                    if (!processorsStatus[processorNumbers.get(readyTasks.get(i))]) {
                        processorsStatus[processorNumbers.get(readyTasks.get(i))] =
true;
```

```
                        processorsTask[processorNumbers.get(readyTasks.get(i))] =
readyTasks.get(i);
                    }
                }
            }
            for (int i = 0; i < processorCount; i++) {
                if (processorsStatus[i]) {
                    vertexWeight[processorsTask[i]]--;
                    if (vertexWeight[processorsTask[i]] == 0) {
                        for (int j = 0; j <
connectionMatrix[processorsTask[i]].length; j++) {
                            if (connectionMatrix[processorsTask[i]][j] > 0) {
                                if (processorNumbers.get(j) != i) {
                                    ArrayList<Integer> way = getWay(i,
processorNumbers.get(j));
                                    int channelNumber = -1;
                                    int k = 0;
                                    while ((channelNumber < 0) && (k < channelCount))
{
                                        if (((channelProcessorFrom[k] == way.get(0))
&& (channelProcessorTo[k] == way.get(1))) ||
                                                ((channelProcessorTo[k] == way.get(0))
&& (channelProcessorFrom[k] == way.get(1)))) {
                                            channelNumber = k;
                                        }
                                        k++;
                                    }

channelFromQueue.get(channelNumber).add(processorsTask[i]);
                                    channelToQueue.get(channelNumber).add(j);
                                    channelWayQueue.get(channelNumber).add(way);

channelWeightQueue.get(channelNumber).add(connectionMatrix[processorsTask[i]][j]);
                                    startNextTact.get(channelNumber).add(true);
                                }
                                else {
                                    connectionMatrix[processorsTask[i]][j] = 0;
                                }
                            }
                        }
                        processorsStatus[i] = false;
                    }
                }
            }
            for (int i = 0; i < channelCount; i++) {
                if (channelsStatus[i]) {
                    channelTaskWeight[i]--;
                    if (!((channelProcessorTo[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)) ||
                            (channelProcessorFrom[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)))) {
                        if (!isAlreadySend[i]) {
                            isAlreadySend[i] = true;
                            ArrayList<Integer> nextTaskWay = new ArrayList<Integer>();
                            for (int l = 1; l < channelTaskWay.get(i).size(); l++) {
                                nextTaskWay.add(channelTaskWay.get(i).get(l));
                            }
                            int channelNumber = -1;
                            int k = 0;
                            while ((channelNumber < 0) && (k < channelCount)) {
                                if (((channelProcessorFrom[k] == nextTaskWay.get(0))
&&
                                        (channelProcessorTo[k] == nextTaskWay.get(1)))
||
                                        ((channelProcessorTo[k] == nextTaskWay.get(0))
&&
                                                (channelProcessorFrom[k] ==
nextTaskWay.get(1)))) {
                                    channelNumber = k;
                                }
                                k++;
```

```
                            }
                            channelFromQueue.get(channelNumber).add(channelFrom[i]);
                            channelToQueue.get(channelNumber).add(channelTo[i]);
                            channelWayQueue.get(channelNumber).add(nextTaskWay);

channelWeightQueue.get(channelNumber).add(channelTaskWeightBackup[i]);
                            channelTaskWeightBackup[i] = 0;
                            startNextTact.get(channelNumber).add(true);
                        }
                    }
                    if (channelTaskWeight[i] == 0) {
                        if ((channelProcessorTo[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)) ||
                                (channelProcessorFrom[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1))) {
                            connectionMatrix[channelFrom[i]][channelTo[i]] = 0;
                        }
                        channelsStatus[i] = false;
                    }
                }
                else {
                    if (!channelWeightQueue.get(i).isEmpty()) {
                        if (!startNextTact.get(i).get(0)) {
                            isAlreadySend[i] = false;
                            startNextTact.get(i).remove(0);
                            channelFrom[i] = channelFromQueue.get(i).get(0);
                            channelFromQueue.get(i).remove(0);
                            channelTo[i] = channelToQueue.get(i).get(0);
                            channelToQueue.get(i).remove(0);
                            channelTaskWay.set(i, channelWayQueue.get(i).get(0));
                            channelWayQueue.get(i).remove(0);
                            channelTaskWeight[i] = channelWeightQueue.get(i).get(0);
                            channelTaskWeightBackup[i] =
channelWeightQueue.get(i).get(0);
                            channelWeightQueue.get(i).remove(0);
                            channelsStatus[i] = true;
                            channelTaskWeight[i]--;
                            if (!((channelProcessorTo[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)) ||
                                    (channelProcessorFrom[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)))) {
                                if (!isAlreadySend[i]) {
                                    isAlreadySend[i] = true;
                                    ArrayList<Integer> nextTaskWay = new
ArrayList<Integer>();
                                    for (int l = 1; l < channelTaskWay.get(i).size();
l++) {
                                        nextTaskWay.add(channelTaskWay.get(i).get(l));
                                    }
                                    int channelNumber = -1;
                                    int k = 0;
                                    while ((channelNumber < 0) && (k < channelCount))
{
                                        if (((channelProcessorFrom[k] ==
nextTaskWay.get(0)) &&
                                                (channelProcessorTo[k] ==
nextTaskWay.get(1))) ||
                                                ((channelProcessorTo[k] ==
nextTaskWay.get(0)) &&
                                                        (channelProcessorFrom[k] ==
nextTaskWay.get(1)))) {
                                            channelNumber = k;
                                        }
                                        k++;
                                    }

channelFromQueue.get(channelNumber).add(channelFrom[i]);

channelToQueue.get(channelNumber).add(channelTo[i]);

channelWayQueue.get(channelNumber).add(nextTaskWay);
```

```
channelWeightQueue.get(channelNumber).add(channelTaskWeightBackup[i]);
                                    channelTaskWeightBackup[i] = 0;
                                    startNextTact.get(channelNumber).add(true);
                                }
                            }
                            if (channelTaskWeight[i] == 0) {
                                if ((channelProcessorTo[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)) ||
                                        (channelProcessorFrom[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1))) {
                                    connectionMatrix[channelFrom[i]][channelTo[i]] =
0;
                                }
                                channelsStatus[i] = false;
                            }
                        }
                    }
                }
            }
            time++;
        }
        return time;
    }

    private void fillTable(ArrayList<Integer> processorNumbers, int[][]
originalConnectionMatrix, int[] originalVertexWeight,
                           String[] vertexNames, GanttTableModel model) {
        int[][] connectionMatrix = new int[originalConnectionMatrix.length][];
        for (int i = 0; i < connectionMatrix.length; i++) {
            connectionMatrix[i] = new int[originalConnectionMatrix[i].length];
            for (int j = 0; j < connectionMatrix[i].length; j++) {
                connectionMatrix[i][j] = originalConnectionMatrix[i][j];
            }
        }
        int[] vertexWeight = new int[originalVertexWeight.length];
        for (int i = 0; i < vertexWeight.length; i++) {
            vertexWeight[i] = originalVertexWeight[i];
        }
        int time = 0;
        boolean[] processorsStatus = new boolean[processorCount];
        for (boolean e : processorsStatus) {
            e = false;
        }
        int[] processorsTask = new int[processorCount];
        for (int e : processorsTask) {
            e = -1;
        }
        int[] channelProcessorFrom = new int[channelCount];
        int[] channelProcessorTo = new int[channelCount];
        int fromProcessor = 0;
        int tempCounter = 0;
        for (int j = 0; j < channelCount; j++) {
            channelProcessorFrom[j] = fromProcessor;
            channelProcessorTo[j] = j + 1;
            tempCounter++;
            if (tempCounter > 1) {
                tempCounter = 0;
                fromProcessor++;
            }
        }
        boolean[] channelsStatus = new boolean[channelCount];
        for (boolean e : channelsStatus) {
            e = false;
        }
        int[] channelFrom = new int[channelCount];
        for (int e : channelFrom) {
            e = -1;
        }
        int[] channelTo = new int[channelCount];
        for (int e : channelTo) {
```

```
            e = -1;
        }
        ArrayList<ArrayList<Boolean>> startNextTact = new
ArrayList<ArrayList<Boolean>>();
        boolean[] isAlreadySend = new boolean[channelCount];
        int[] channelTaskWeight = new int[channelCount];
        int[] channelTaskWeightBackup = new int[channelCount];
        ArrayList<ArrayList<Integer>> channelTaskWay = new
ArrayList<ArrayList<Integer>>();
        ArrayList<ArrayList<Integer>> channelFromQueue = new
ArrayList<ArrayList<Integer>>();
        ArrayList<ArrayList<Integer>> channelToQueue = new
ArrayList<ArrayList<Integer>>();
        ArrayList<ArrayList<ArrayList<Integer>>> channelWayQueue = new
ArrayList<ArrayList<ArrayList<Integer>>>();
        ArrayList<ArrayList<Integer>> channelWeightQueue = new
ArrayList<ArrayList<Integer>>();
        for (int i = 0; i < channelCount; i++) {
            startNextTact.add(new ArrayList<Boolean>());
            isAlreadySend[i] = false;
            channelTaskWeight[i] = -1;
            channelTaskWay.add(new ArrayList<Integer>());
            channelFromQueue.add(new ArrayList<Integer>());
            channelToQueue.add(new ArrayList<Integer>());
            channelWayQueue.add(new ArrayList<ArrayList<Integer>>());
            channelWeightQueue.add(new ArrayList<Integer>());
        }
        while (!isAllTasksDone(vertexWeight)) {
            for (int i = 0; i < channelCount; i++) {
                for (int j = 0; j < startNextTact.get(i).size(); j++) {
                    startNextTact.get(i).set(j, false);
                }
            }
            String[] newRow = new String[processorCount + channelCount + 1];
            newRow[0] = String.valueOf(time);
            ArrayList<Integer> readyTasks = getReadyTasks(connectionMatrix);
            for (int i = 0; i < readyTasks.size(); i++) {
                if (vertexWeight[readyTasks.get(i)] > 0) {
                    if (!processorsStatus[processorNumbers.get(readyTasks.get(i))]) {
                        processorsStatus[processorNumbers.get(readyTasks.get(i))] =
true;
                        processorsTask[processorNumbers.get(readyTasks.get(i))] =
readyTasks.get(i);
                    }
                }
            }
            for (int i = 0; i < processorCount; i++) {
                if (processorsStatus[i]) {
                    newRow[i + 1] = vertexNames[processorsTask[i]];
                    vertexWeight[processorsTask[i]]--;
                    if (vertexWeight[processorsTask[i]] == 0) {
                        for (int j = 0; j <
connectionMatrix[processorsTask[i]].length; j++) {
                            if (connectionMatrix[processorsTask[i]][j] > 0) {
                                if (processorNumbers.get(j) != i) {
                                    ArrayList<Integer> way = getWay(i,
processorNumbers.get(j));
                                    int channelNumber = -1;
                                    int k = 0;
                                    while ((channelNumber < 0) && (k < channelCount))
{
                                        if (((channelProcessorFrom[k] == way.get(0))
&& (channelProcessorTo[k] == way.get(1))) ||
                                                ((channelProcessorTo[k] == way.get(0))
&& (channelProcessorFrom[k] == way.get(1)))) {
                                            channelNumber = k;
                                        }
                                        k++;
                                    }

channelFromQueue.get(channelNumber).add(processorsTask[i]);
```

```
                                    channelToQueue.get(channelNumber).add(j);
                                    channelWayQueue.get(channelNumber).add(way);
channelWeightQueue.get(channelNumber).add(connectionMatrix[processorsTask[i]][j]);
                                    startNextTact.get(channelNumber).add(true);
                                }
                                else {
                                    connectionMatrix[processorsTask[i]][j] = 0;
                                }
                            }
                        }
                        processorsStatus[i] = false;
                    }
                }
            }
            for (int i = 0; i < channelCount; i++) {
                if (channelsStatus[i]) {
                    StringBuilder builder = new StringBuilder();
                    builder.append(vertexNames[channelFrom[i]]);
                    builder.append(" -> ");
                    builder.append(vertexNames[channelTo[i]]);
                    newRow[i + processorCount + 1] = builder.toString();
                    channelTaskWeight[i]--;
                    if (!((channelProcessorTo[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)) ||
                            (channelProcessorFrom[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)))) {
                        if (!isAlreadySend[i]) {
                             isAlreadySend[i] = true;
                            ArrayList<Integer> nextTaskWay = new ArrayList<Integer>();
                            for (int l = 1; l < channelTaskWay.get(i).size(); l++) {
                                nextTaskWay.add(channelTaskWay.get(i).get(l));
                            }
                            int channelNumber = -1;
                            int k = 0;
                            while ((channelNumber < 0) && (k < channelCount)) {
                                if (((channelProcessorFrom[k] == nextTaskWay.get(0))
&&
                                        (channelProcessorTo[k] == nextTaskWay.get(1)))
||
                                        ((channelProcessorTo[k] == nextTaskWay.get(0))
&&
                                                (channelProcessorFrom[k] ==
nextTaskWay.get(1)))) {
                                    channelNumber = k;
                                }
                                k++;
                            }
                            channelFromQueue.get(channelNumber).add(channelFrom[i]);
                            channelToQueue.get(channelNumber).add(channelTo[i]);
                            channelWayQueue.get(channelNumber).add(nextTaskWay);
channelWeightQueue.get(channelNumber).add(channelTaskWeightBackup[i]);
                            channelTaskWeightBackup[i] = 0;
                            startNextTact.get(channelNumber).add(true);
                        }
                    }
                    if (channelTaskWeight[i] == 0) {
                        if ((channelProcessorTo[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)) ||
                                (channelProcessorFrom[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1))) {
                            connectionMatrix[channelFrom[i]][channelTo[i]] = 0;
                        }
                        channelsStatus[i] = false;
                    }
                }
                else {
                    if (!channelWeightQueue.get(i).isEmpty()) {
                        if (!startNextTact.get(i).get(0)) {
                            isAlreadySend[i] = false;
```

```
                            channelFrom[i] = channelFromQueue.get(i).get(0);
                            channelFromQueue.get(i).remove(0);
                            channelTo[i] = channelToQueue.get(i).get(0);
                            channelToQueue.get(i).remove(0);
                            channelTaskWay.set(i, channelWayQueue.get(i).get(0));
                            channelWayQueue.get(i).remove(0);
                            channelTaskWeight[i] = channelWeightQueue.get(i).get(0);
                            channelTaskWeightBackup[i] =
channelWeightQueue.get(i).get(0);
                            channelWeightQueue.get(i).remove(0);
                            channelsStatus[i] = true;
                            StringBuilder builder = new StringBuilder();
                            builder.append(vertexNames[channelFrom[i]]);
                            builder.append(" -> ");
                            builder.append(vertexNames[channelTo[i]]);
                            newRow[i + processorCount + 1] = builder.toString();
                            channelTaskWeight[i]--;
                            if (!((channelProcessorTo[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)) ||
                                    (channelProcessorFrom[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)))) {
                                if (!isAlreadySend[i]) {
                                    isAlreadySend[i] = true;
                                    ArrayList<Integer> nextTaskWay = new
ArrayList<Integer>();
                                    for (int l = 1; l < channelTaskWay.get(i).size();
l++) {
                                        nextTaskWay.add(channelTaskWay.get(i).get(l));
                                    }
                                    int channelNumber = -1;
                                    int k = 0;
                                    while ((channelNumber < 0) && (k < channelCount))
{
                                        if (((channelProcessorFrom[k] ==
nextTaskWay.get(0)) &&
                                                (channelProcessorTo[k] ==
nextTaskWay.get(1))) ||
                                                ((channelProcessorTo[k] ==
nextTaskWay.get(0)) &&
                                                        (channelProcessorFrom[k] ==
nextTaskWay.get(1)))) {
                                            channelNumber = k;
                                        }
                                        k++;
                                    }

channelFromQueue.get(channelNumber).add(channelFrom[i]);

channelToQueue.get(channelNumber).add(channelTo[i]);

channelWayQueue.get(channelNumber).add(nextTaskWay);

channelWeightQueue.get(channelNumber).add(channelTaskWeightBackup[i]);
                                    channelTaskWeightBackup[i] = 0;
                                    startNextTact.get(channelNumber).add(true);
                                }
                            }
                            if (channelTaskWeight[i] == 0) {
                                if ((channelProcessorTo[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1)) ||
                                        (channelProcessorFrom[i] ==
channelTaskWay.get(i).get(channelTaskWay.get(i).size() - 1))) {
                                    connectionMatrix[channelFrom[i]][channelTo[i]] =
0;
                                }
                                channelsStatus[i] = false;
                            }
                        }
                    }
                }
            }
```

```
            model.addRow(newRow);
            time++;
        }
    }

    private void sortPopulation(ArrayList<int[]> population, ArrayList<Integer>
populationTime) {
        for (int i = 0; i < populationTime.size() - 1; i++) {
            int minTimeIndex = i;
            for (int j = i + 1; j < populationTime.size(); j++) {
                if (populationTime.get(j) < populationTime.get(minTimeIndex)) {
                    minTimeIndex = j;
                }
            }
            if (minTimeIndex != i) {
                int tempTime = populationTime.get(i);
                int[] tempPopulation = population.get(i);
                populationTime.set(i, populationTime.get(minTimeIndex));
                population.set(i, population.get(minTimeIndex));
                populationTime.set(minTimeIndex, tempTime);
                population.set(minTimeIndex, tempPopulation);
            }
        }
    }

    private void removeClones(ArrayList<int[]> population, ArrayList<Integer>
populationTime) {
        for (int i = 0; i < population.size() - 1; i++) {
            for (int j = i + 1; j < population.size(); j++) {
                boolean isClone = true;
                int k = 0;
                while ((isClone) && (k < population.get(j).length)) {
                    if (population.get(j)[k] != population.get(i)[k]) {
                        isClone = false;
                    }
                    k++;
                }
                if (isClone) {
                    population.remove(j);
                    populationTime.remove(j);
                }
            }
        }
    }

    public GanttTableModel loadGraph(int[][] connectionMatrix, int[] vertexWeights,
String[] vertexNames) {
        GanttTableModel tableModel = new GanttTableModel(processorCount + channelCount
+ 1);
        String[] columnNamesRow = new String[processorCount + channelCount + 1];
        columnNamesRow[0] = "Такт";
        int i = 1;
        for (int j = 0; j < processorCount; j++) {
            StringBuilder builder = new StringBuilder();
            builder.append("Процессор ");
            builder.append(String.valueOf(j));
            columnNamesRow[i] = builder.toString();
            i++;
        }
        int fromProcessor = 0;
        int tempCounter = 0;
        for (int j = 0; j < channelCount; j++) {
            StringBuilder builder = new StringBuilder();
            builder.append("Канал ");
            builder.append(String.valueOf(fromProcessor));
            builder.append("-");
            builder.append(String.valueOf(j + 1));
            columnNamesRow[i] = builder.toString();
            i++;
            tempCounter++;
            if (tempCounter > 1) {
```

```
                tempCounter = 0;
                fromProcessor++;
            }
        }
        tableModel.addColumnNamesRow(columnNamesRow);
        ArrayList<Integer> theBestGenotype = null;
        int theBestTime = Integer.MAX_VALUE;
        for (int x = 0; x < STEP_COUNT; x++) {
            ArrayList<int[]> population = new ArrayList<int[]>();
            ArrayList<Integer> populationTime = new ArrayList<Integer>();
            for (int j = 0; j < populationSize; j++) {
                int[] newGenotype = new int[vertexNames.length];
                for (int k = 0; k < newGenotype.length; k++) {
                    int randomValue = (int) (Math.random() * 1000);
                    int step = 1000 / processorCount;
                    int processorNumber = (randomValue / step);
                    while (processorNumber >= processorCount) {
                        processorNumber--;
                    }
                    newGenotype[k] = processorNumber;
                }

                population.add(newGenotype);
                ArrayList<Integer> newGenotypeList = new ArrayList<Integer>();
                for (int k = 0; k < newGenotype.length; k++) {
                    newGenotypeList.add(newGenotype[k]);
                }
                populationTime.add(getExecutionTime(newGenotypeList, connectionMatrix,
vertexWeights));
            }
            sortPopulation(population, populationTime);
            int[] bestGenotype = population.get(0);
            int bestTime = populationTime.get(0);
            int bestGenotypeNotChangedCount = 0;
            do {
                for (int j = 0; j < populationSize; j++) {
                    int randomValue = (int) (Math.random() * 1000);
                    int populationStep = 1000 / populationSize;
                    int crossIndex = randomValue / populationStep;
                    if (crossIndex == j) {
                        if (j > 0) {
                            crossIndex--;
                        } else {
                            crossIndex++;
                        }
                    }
                    while (crossIndex >= population.size()) {
                        crossIndex--;
                    }
                    int[] newGenotype = new int[population.get(j).length];
                    for (int k = 0; k < population.get(j).length; k++) {
                        double randomValueDounle = Math.random();
                        if (randomValueDounle >= CROSS_PROBABILITY) {
                            newGenotype[k] = population.get(j)[k];
                        } else {
                            newGenotype[k] = population.get(crossIndex)[k];
                        }
                    }
                    population.add(newGenotype);
                    ArrayList<Integer> newGenotypeList = new ArrayList<Integer>();
                    for (int k = 0; k < newGenotype.length; k++) {
                        newGenotypeList.add(newGenotype[k]);
                    }
                    populationTime.add(getExecutionTime(newGenotypeList,
connectionMatrix, vertexWeights));
                }
                for (int j = 0; j < populationSize; j++) {
                    int[] newGenotype = new int[population.get(j).length];
                    for (int k = 0; k < population.get(j).length; k++) {
                        double randomValue = Math.random();
                        if (randomValue < MUTATION_PROBABILITY) {
```

```
                            int randomValueInt = (int) (Math.random() * 1000);
                            int step = 1000 / processorCount;
                            int processorNumber = (randomValueInt / step);
                            while (processorNumber >= processorCount) {
                                processorNumber--;
                            }
                            newGenotype[k] = processorNumber;
                        } else {
                            newGenotype[k] = population.get(j)[k];
                        }
                    }
                    population.add(newGenotype);
                    ArrayList<Integer> newGenotypeList = new ArrayList<Integer>();
                    for (int k = 0; k < newGenotype.length; k++) {
                        newGenotypeList.add(newGenotype[k]);
                    }
                    populationTime.add(getExecutionTime(newGenotypeList,
connectionMatrix, vertexWeights));
                }
                removeClones(population, populationTime);
                sortPopulation(population, populationTime);
                for (int j = population.size() - 1; j >= populationSize; j--) {
                    population.remove(j);
                    populationTime.remove(j);
                }
                boolean isBestChanged = false;
                int j = 0;
                while ((!isBestChanged) && (j < population.get(0).length)) {
                    if (population.get(0)[j] != bestGenotype[j]) {
                        isBestChanged = true;
                    }
                    j++;
                }
                if (isBestChanged) {
                    bestGenotype = population.get(0);
                    bestTime = populationTime.get(0);
                    bestGenotypeNotChangedCount = 0;
                } else {
                    bestGenotypeNotChangedCount++;
                }
            }
            while (bestGenotypeNotChangedCount < BEST_GENOTYPE_NOT_CHANGED_MAX_COUNT);
            if (bestTime < theBestTime) {
                ArrayList<Integer> bestGenotypeList = new ArrayList<Integer>();
                for (int k = 0; k < bestGenotype.length; k++) {
                    bestGenotypeList.add(bestGenotype[k]);
                }
                theBestGenotype = bestGenotypeList;
                theBestTime = bestTime;
            }
        }
        fillTable(theBestGenotype, connectionMatrix, vertexWeights, vertexNames,
tableModel);
        return tableModel;
    }

}
```