МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра дискретного анализа и исследования операций

Направление подготовки: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

# Генетический алгоритм для задачи балансировки нагрузки на серверы

Дорожко Антон Владимирович

**«К защите допущена»**

Заведующий кафедрой,

д.ф.-м.н., профессор

Береснев В.Л. /……..

«……»……20…г.

**Научный руководитель**

в. н. с., ИМ СО РАН,

д.ф.-м.н., профессор

Кочетов Ю.А. /……..

«……»……20…г.

Дата защиты: «……»……20…г.

Новосибирск, 2015г.

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра дискретного анализа и исследования операций

Направление подготовки: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

УТВЕРЖДАЮ

Зав. Кафедрой Береснев В.Л.
(фамилия, И., О.)
…….
(подпись, МП)
«……»……20…г.

**ЗАДАНИЕ**

**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

Студенту Дорожко Антону Владимировичу
Тема: Генетический алгоритм для задачи балансировки нагрузки на серверы
Исходные данные (или цель работы): разработка и исследование генетического алгоритма для задачи балансировки нагрузки на серверы

Структурные части работы: реализация генетического алгоритма для задачи балансировки нагрузки на серверы на C++; исследование различных вариантов алгоритма; тестирование на данных, предоставленных компанией, занимающейся хостингом; исследование алгоритма при значительном увеличении размерности входных данных.

Научный руководитель
в. н. с., ИМ СО РАН,

д.ф.-м.н., профессор
Кочетов Ю.А./……..
(фамилия , И., О.) / (подпись)
«…»……20…г.

Задание принял к исполнению
Дорожко А.В./……..

(ФИО студента) / (подпись)
«…»……20…г.

Содержание

## ВВЕДЕНИЕ

Компании, предоставляющие хостинг, заинтересованы в том, чтобы вычислительные мощности их серверов использовались максимально полно. В связи с динамическим изменением количества запросов к информации на сервере, возникает желание уметь балансировать эту нагрузку.

Рассмотрим задачу балансировки подробнее. Пусть имеется набор серверов, на каждом из которых размещены диски. На каждом диске размещено несколько сайтов с разнородным контентом. При любом обращении пользователя к сайтам создается нагрузка на вычислительные мощности сервера. Конкретнее, эта нагрузка выражается в потреблении таких ресурсов как ЦП, ОЗУ и др. Известна статистика обращений пользователей по каждому диску в течение планового периода, что позволяет определить нагрузку на отдельно взятый сервер в единицу времени по каждому ресурсу. Если нагрузка по каждому ресурсу не превосходит заданного порога, то сервер находится в рабочем режиме. Иначе сервер будет испытывать повышенную нагрузку, вследствие чего будут возникать задержки на запросы пользователей.

Кроме самой статистики обращений к сайтам на дисках предоставляется прогноз по росту нагрузки на каждый диск. Вводится три типа дисков: маленькие, потенциально большие и большие. Маленькие диски не накладывают никаких дополнительных ограничений. Потенциально большие диски – это диски, для которых предсказан значительный рост нагрузки, и с высокой вероятностью они станут большими. Большие диски – диски, занимающие значительные ресурсы, и для предоставления качественного уровня обслуживания пользователей ресурсов, размещенных на этих дисках, требуется предоставлять выделенный сервер. Задача классификации дисков не рассматривается в данной работе. Результат решения задачи является частью входных данных.

Существует несколько вариантов балансировки нагрузки. В данной работе мы будем строить алгоритм для оптимизации нагрузки путем перемещения (переключения) дисков с сервера на сервер. Такое перемещение требует определенных вычислительных затрат (далее накладные расходы). Предполагается, что для каждого диска известны накладные расходы по каждому ресурсу на «изъятие» с, «вставку» на любой сервер. Начальное расположение дисков на серверах задано.

Задача в текущей постановке является достаточно новой, но уже имеются значительные результаты. Для данной задачи разработана математическая модель в терминах частично-целочисленного линейного программирования[11]. Показана NP-

трудность задачи и проведены численные эксперименты с применением коммерческого программного обеспечения для решения задач линейного программирования IMB ILOG CPLEX[1] [11]. Для решения предложенной задачи реализован приближенный алгоритм, использующий метод локального поиска по окрестности, которая является задачей о назначениях [5]. Для решения похожей задачи, но без классификации дисков, реализованы метаэвристики: метод имитации отжига[9] и поиск с запретами [1, 2]. Приведены варианты организации локального поиска с окрестностями Move, Swap, Assign. Проведены численные эксперименты на тестовых данных. Автор диплома описывает применение генетического алгоритма (ГА) [8] для решения задачи балансировки нагрузки на серверы. Работы [2, 8] показывают сравнимые результаты с небольшим преимуществом по качеству решений и времени у поиска с запретами.

В дипломной работе исследуется применение ГА для решения задачи балансировки нагрузки на серверы с учетом классификации дисков по нагрузке.

Основные задачи:

- разработка ГА для задачи балансировки
- разработка методов локального поиска для задачи балансировки
- исследование улучшения алгоритма за счет введения различных окрестностей и исследования влияния параметров алгоритма
- тестирование на реальных данных, исследование поведения алгоритма при значительном росте размерности входных данных

[1] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

## 1 Математическая постановка задачи

*Исходные данные для решения задачи*:

$T$ — интервал планирования;

$R$ — множество характеристик нагрузки;

$c_{drt}$ — нагрузка диска $d$ по характеристике $r$ в момент $t$;

$\overline{c}_{sr}$ — пороговая нагрузка сервера $s$ по характеристике $r$;

$x_{ds}^0$ — начальное распределение дисков по серверам;

$b_{sdr}^w (b_{sdr}^e)$ — накладные расходы по характеристике $r$ на подключение (изъятие) диска $d$ в сервер $s$;

$B_{sr}$ — предельно допустимые накладные расходы по характеристике $r$ для сервера $s$;

$D_0$ - множество больших дисков;

$D_1$ - множество потенциально больших дисков.

*Переменные задачи*: распределение дисков по серверам

$$x_{ds} = \begin{cases} 1, \text{ если } d \text{ ставится на сервер } s \\ 0, \text{ в противном случае} \end{cases}$$

$y_{str}$ — перегрузка на сервере $s$ в момент времени $t$ по характеристике $r$.

Дополнительные обозначения:

$S_0, S_1$ - сервера содержащие большие и потенциально большие диски из множеств $D_0, D_1$ соответственно

С использованием введенных обозначений задача балансировки нагрузки на серверы с большими и потенциально большими сайтами может быть записана в виде следующей задачи частично–целочисленного линейного программирования:

$$\min \sum_{s \in S} \sum_{t \in T} \sum_{r \in R} y_{str} \tag{1}$$

при следующих ограничениях:

$$y_{str} = \max \left\{ \sum_{d \in D} c_{drt} x_{ds} - \bar{c}_{sr}, 0 \right\} \text{ для всех } s \in S,\ t \in T,\ r \in R, \tag{2}$$

$$\sum_{s \in S} x_{ds} = 1 \quad \text{для всех } d \in D, \tag{3}$$

$$\sum_{d \in D} b^{w}_{sdr} x_{ds} (1 - x^{0}_{ds}) + \sum_{d \in D} b^{e}_{sdr} (1 - x_{ds}) x^{0}_{ds} \le B_{sr} \text{ для всех } s \in S,\ r \in R, \tag{4}$$

$$y_{str} \ge 0,\ x_{ds} \in \{0,1\} \text{ для всех } d \in D,\ s \in S,\ t \in T,\ r \in R. \tag{5}$$

$$\sum_{d \in D_1} x_{ds} \le 1 \quad \text{для всех } s \in S \setminus S_0, \tag{6}$$

$$x_{ds} = x^{0}_{ds} \text{ для всех } d \in D_0, s \in S_0, \tag{7}$$

$$x_{ds} = 0 \text{ для всех } d \notin D_0, s \in S_0, \tag{8}$$

Целевая функция (1) задает суммарную перегрузку серверов на всем интервале планирования. Равенство (2) позволяет определить эту перегрузку для каждого сервера в каждый момент времени по каждой характеристике. Равенство (3) гарантирует ровно один сервер для каждого диска. Неравенства (4) определяет ограничения на накладные расходы. Условие (5) задает тип переменных: булевы переменные $x_{ds}$ и непрерывные неотрицательные переменные $y_{str}$. Неравенство (6) обеспечивает запрет на размещение нескольких потенциально больших сайтов на одном сервере.

Равенства (7) –(8) обеспечивают разгрузку серверов с большими сайтами. Равенство (7) гарантирует, что диски из множества $D_0$ не будут перемещаться. Равенство (8) требует убрать с серверов из множества $S_0$ все диски, не принадлежащие множеству $D_0$.

## 2 Метод решения задачи балансировки нагрузки с классами

Предлагаемый далее метод решения задачи (1) при ограничениях (2) – (8) состоит из двух этапов. На первом этапе производится разгрузка серверов из множества $S_0$ (серверов с большими дисками), то есть находится некоторое *полудопустимое* решение $\overline{x}_{ds}$, удовлетворяющее всем ограничениям задачи, кроме неравенства (6) для потенциально больших веб-сайтов. На втором этапе это решение используется в качестве начального распределения дисков для балансировки нагрузки на серверы из множества $S \setminus S_0$ с учетом неравенства (6). Для этой цели неравенство (6) заменяется штрафом $W * \max\left[0, \sum_{d \in D_1} x_{ds} - 1\right]$ в целевую функцию и решается вспомогательная задача

$$\min \sum_{s \in S \setminus S_0} \left( \sum_{t \in T} \sum_{r \in R} y_{str} + W * \max\left[0, \sum_{d \in D_1} x_{ds} - 1\right] \right) \qquad (9)$$

с учетом ограничений (2), (3), (5) и дополнительного нового ограничения

$$\sum_{d \in D} b^w_{sdr} x_{ds} (1 - x^0_{ds}) + \sum_{d \in D} b^e_{sdr} (1 - x_{ds}) x^0_{ds} \le B_{sr} - \sum_{d \in D} b^w_{sdr} \overline{x}_{ds} (1 - x^0_{ds}), \text{ для } s \in S / S_0, r \in R$$

где матрица $\bar{x}_{ds}$ задает распределение дисков по серверам после выполнения первого этапа. Последнее ограничение отличается от (4), так как часть накладных расходов на подключение дисков уже потрачена при разгрузке серверов из $S_0$. Для решения этой задачи предлагается применить метаэвристики [3, 6].

### 2.1 Описание алгоритма решения задачи

Предлагаемый двухэтапный метод основан на сведении рассматриваемой задачи к уже исследованной задаче балансировки нагрузки на серверы[8]. Первый этап фактически состоит в разгрузке серверов с большими сайтами и подготовке исходных данных для второго этапа. На втором этапе применяются методы локального поиска, эффективность которых подтверждена вычислительными экспериментами. Дополнительное слагаемое в целевой функции $W * \max\left[0, \sum_{d \in D_1} x_{ds} - 1\right]$ играет роль штрафа за нарушение ограничений (6). Выбор коэффициента $W$ достаточно большим числом гарантирует получение допустимого решения. Такой подход широко применяется в методах оптимизации и носит название метода штрафных функций [4, 15, 17].

Целью первого этапа предложенного метода является получение полудопустимого решения $\bar{x}_{ds}$ для разгрузки серверов из множества $S_0$. Следует отметить, что поиск полудопустимого решения является NP-полной задачей из-за наличия ограничения (4) на подключение дисков. Даже наличие одной характеристики нагрузки ($|R|=1$) требует размещения всех дисков из множества $D \setminus D_0$ на серверах $S \setminus S_0$ в рамках ограничения (8). Следовательно, данная задача соотносится с классической задачей упаковки в контейнеры, являющейся NP-трудной. Это является основанием применить для ее решения алгоритмы локального поиска [15, 20].

Решение задачи первого этапа требует предварительную проверку совместности системы ограничений (2) – (8) и, в частности, условия (4) для серверов из множества $S_0$. Так как все диски, за исключением содержащихся в множестве $D_0$, должны быть удалены из серверов множества $D_0$, то для каждого $s \in S_0$ можно вычислить накладные расходы на изъятие дисков (второе слагаемое в левой части неравенства (4)). Если для сервера $s \in S_0$ эта величина превышает значение $B_{sr}$ хотя бы по одному из параметров $r \in R$, то система ограничений (2) – (8) будет считаться несовместной, что потребует корректировку исходных данных задачи. Если при этом для каждой пары $(s,r)$, где $s \in S_0$ и $r \in R$, условие (4) не нарушается при удалении дисков, то это еще не гарантирует существования допустимого решения первого этапа, потому что на серверах, на которые будут распределены диски могут быть превышены ограничения на накладные расходы при вставке всех перемещаемых дисков. Для нахождения так называемого *допустимого* решения необходимо решить следующую подзадачу.

Пусть $x^1_{ds}$ - решение, полученное после перемещения всех дисков из множества $D \setminus D_0$ (т.е. все диски кроме больших) на сервера из множества $S \setminus S_0$. На этом шаге мы можем нарушать ограничение (4). Это решение может быть получено различными способами: жадным или рандомизированным алгоритмами. Пусть $B(x^1_{ds})$ означает суммарное нарушения ограничения, заданного неравенством (4) для этого решения, представленную как

$$B(x^1_{ds}) = \sum_{s \in S \setminus S_0} \sum_{r \in R} \max\left(0, \sum_{d \in D} b^w_{sdr} x^1_{ds} (1 - x^0_{ds}) - B_{sr}\right) \qquad (10)$$

Если $B(x_{ds}^1) = 0$, тогда решение первого этапа допустимо. Иначе необходимо перераспределить диски, для этого достаточно использовать локальный поиск по окрестности Move. Если не удается решить задачу первого этапа, то это означает, что даже если допустимое решение существует, то найти его достаточно трудно, а это в свою очередь свидетельствует о том, что для текущих прогнозов нагрузки на диски (т.е. распределения по классам) серверная инфраструктура обладает недостаточным количеством ресурсов и необходимо либо добавлять дополнительные сервера, либо пересматривать прогнозы.

Допустимое решение задачи для первого этапа полагаем начальным решением для второго этапа. Но чтобы использовать генетический алгоритм необходимо изменить входные данные следующим образом: вспомним, что $B_{sr}$ — предельно допустимые накладные расходы по характеристике $r$ для сервера $s$, а $b_{sdr}^w (b_{sdr}^e)$ — накладные расходы по характеристике $r$ на подключение(изъятие) диска $d$ в сервер $s$. Тогда для всех серверов $s \in S_0$, полагаем $b_{sdr}^w > B_{sr} \ \forall r \in R, \forall d \notin s$ в решении первого этапа Таким образом, запретив алгоритму вставлять диски в сервера с большим диском. Измененное условие и начальное решение, полученное на первом этапе, передаем на вход генетическому алгоритму.

## 2.2 Построение генетического алгоритма

Генетические алгоритмы были предложены Джоном Холландом [19] в 1970-ых (Университет Мичиган, США), чтобы понять адаптивные процессы в естественных системах. Потом они были применены к проблемам оптимизации и машинного обучения в 1980-ых. На сегодняшний день генетические алгоритмы доказали свою конкурентоспособность при решении многих NP-трудных задач, особенно в приложениях, где математические модели имеют сложную структуру и применение стандартных методов типа ветвей и границ, динамического или линейного программирования крайне затруднено [20].

Идея генетических алгоритмов заимствована у живой природы и состоит в организации эволюционного процесса, конечной целью которого является получение оптимального решения сложной комбинаторной задачи. Разработчик генетических алгоритмов выступает в данном случае как "создатель", который должен правильно установить законы эволюции, чтобы достичь этой цели.

Общая схема генетического алгоритма имеет следующий вид:

1. Сгенерировать начальную популяцию
2. Применить **алгоритм локального поиска** к решениям популяции
3. Пока не выполнен **критерий остановки**
   - 3.1. Выбрать решения $x_1, x_2$ , получить новое решения $x' = Crossover(x_1, x_2)$
   - 3.2. Применить к $x'$ **алгоритм локального поиска** и добавить в популяцию
   - 3.3. Выбрать решения $x_1$ , получить новое решения $x'' = Mutation(x_1)$
   - 3.4. Применить к $x''$ **алгоритм локального поиска** и добавить в популяцию
   - 3.5. Применить **стратегию замещения**
4. В качестве ответа взять лучшее решение из популяции

Перед тем как заполнять шаблон ГА, следует определиться с кодированием решения. Удобно представить решение в виде вектора размера $|D|$, где $D$ - множество дисков. Тогда $i-ый$ элемент вектора хранит номер сервера, на котором расположен $i-ый$ диск.

Для того, чтобы каждый раз не пересчитывать целевую функцию заново, в качестве вспомогательной структуры используем вектор перегрузок для каждого сервера. Эта структура ускорит пересчет целевой функций после таких локальных изменений решения, как операторы *Move* и *Swap*, которые описанные в пункте 4.2.

### 2.2.1 Генерация начальной популяции

**Определение**: *Популяция* – множество решений

Первый вопрос, который предстоит решить, это выбор алгоритма генерации начальной популяции. Этот шаг играет важную роль в эффективности метаэвристик, основанных на популяциях. Главным критерием при генерации начальной популяции является диверсификация. Если начальная популяция будет недостаточно диверсифицирована, это может привести к преждевременной сходимости. В алгоритме применялись следующие стратегии: жадная генерация и метод последовательной диверсификации.

Жадная генерация должна быть некоторым образом рандомизированна, иначе разнообразие решений в начальной популяции будет невелико, что приведет к преждевременной сходимости к одинаковым решениям и стагнации популяции.

1. Повторять пока на шаге 3 найден ход улучшающий целевую функцию
   1.1. Выбрать сервер с максимальной перегрузкой
   1.2. Выбрать k случайных дисков из сервера
   1.3. Среди всех перемещений этих k дисков, удовлетворяющих накладным расходам на изъятие и вставку, найти перемещение, дающее наибольшее уменьшение целевой функции и выполнить его.

Жадная генерация позволяет значительно уменьшить значение целевой функции, но в разнообразии решений уступает последовательной диверсификации.

При последовательной диверсификации решения генерируются в такой последовательности, чтобы разнообразие решений было не хуже определенного уровня. В данной задаче уровень разнообразия определяется минимальным расстоянием $\Delta$ по метрике Хэмминга между всеми парами решений начальной популяции. Возьмем текущее множество решений Q (изначально состоит только из начального решения). Применяя несколько раз оператор Move (перемещение одного диска на другой сервер), добавим в Q следующее решение, которое должно быть минимум на расстоянии $\Delta$ от всех решений текущего множества Q. Процесс повторяется, пока не наберется определенное число решений – размер начальной популяции, или после заданного числа итераций. Ограничение по числу итераций нужно, чтобы избежать зацикливания, потому что не всегда наше решение будет удовлетворять ограничению на $\Delta$ и его нужно отбросить.

Если после генерации получили только часть стартовой популяции, используем алгоритм жадной генерации для получения заданного размера популяции.

Минусом этого варианта является относительно высокая вычислительная сложность. Однако мы можем получить популяцию, в которой решения хорошо диверсифицированы. И это позволит лучше исследовать ландшафт пространства решений.

В виду сравнительного анализа методов генерации начальных популяций в [20] и необходимости иметь в начальной популяции хорошее разнообразие решений был выбран метод последовательной диверсификации.

Стратегия замещения определяет, какие решения нужно оставить в популяции. Важно, чтобы эта стратегия сохраняла диверсификацию нашей популяции. Вследствие хорошей диверсификации начальной популяции и правильного выбора оператора скрещивания стало возможным для отбора решений в следующее поколение выбрать простую стратегию элитизма, т.е. оставлять решения с наилучшим значением целевой функции.

### 2.2.2 Алгоритм локального поиска

Большой вклад в успешность таких метаэвристик как поиск с запретами, метод имитации отжига и генетический алгоритм вносит алгоритм локального поиска.

Локальный поиск [5, 10, 13] это простой метаэвристический метод. Начинаем в каком-то стартовом решении. На каждой итерации эвристика заменяет текущее решение на соседа, который не обязательно имеет лучшую целевую функцию (если потребовать условие обязательности улучшения целевой функции, то метод будет называться локальный спуск). Поиск останавливается при определенном условии (например, для локального спуска, когда все соседи в определенной окрестности не лучше текущего решения). Для реализации этого метода необходимо определить стартовое решение, окрестность и стратегию выбора соседа из окрестности.

**Определение**: Окрестность решения $X$ – множество решений, полученных из $X$ одноразовым применением определенной операции.

Введем следующие окрестности:

*Move* — двигаем диск на другой сервер, мощность окрестности равна $O(|D||S|)$ .

*Swap* — меняем один диск на другой, мощность окрестности равна $O(|D|^2)$ .

*Assign* — изъять с каждого сервера по одному диску и распределить их оптимально между серверами, мощность окрестности равна $O(|S|!)$ [16].

$Lin-Kernighan$ – эта окрестность является некоторым обобщением окрестности Move $O(\mathrm{N}|D||S|)$ или Swap $O(\mathrm{N}|D|^2)$, где $N$ - количество шагов по соответствующей окрестности.

В качестве стартового решения будут выступать новые решения в популяции, полученные с помощью операторов скрещивания и мутации.

Окрестность к *Move*

1. Для каждого диска вычислить выигрыш по целевой функции от перемещения на другой сервер
2. Выполнить *Move*, дающий наибольший выигрыш по целевой функции и удовлетворяющий ограничениям (3-4)

Окрестность к *Swap*

1. Для каждой пары дисков вычислить выигрыш по целевой функции от их обмена, если они на разных серверах
2. Выполнить *Swap*, дающий наибольший выигрыш по целевой функции и удовлетворяющий ограничениям (3-4)

Рандомизированная окрестность к Move

1. Выбрать случайным образом $N$ пар сервер-диск
2. Для каждой пары вычислить выигрыш по целевой функции
3. Выполнить Move, дающий наибольший выигрыш по целевой функции и удовлетворяющий ограничениям (3-4)

Рандомизированная окрестность к Swap

1. Выбрать случайным образом $N$ пар дисков $\langle d1, d2\rangle$ *где* $d1 != d2$
2. Для каждой выбранной пары дисков вычислить выигрыш по целевой функции от их обмена
3. Выполнить *Swap*, дающий наибольший выигрыш по целевой функции и удовлетворяющий ограничениям (3-4)

Недостаток полного просмотра окрестностей Move и Swap в неоправданной вычислительной сложности. Затраты времени на просмотр всей окрестности не способствуют получению лучших результатов по сравнению с рандомизированными вариантами (рис.1).

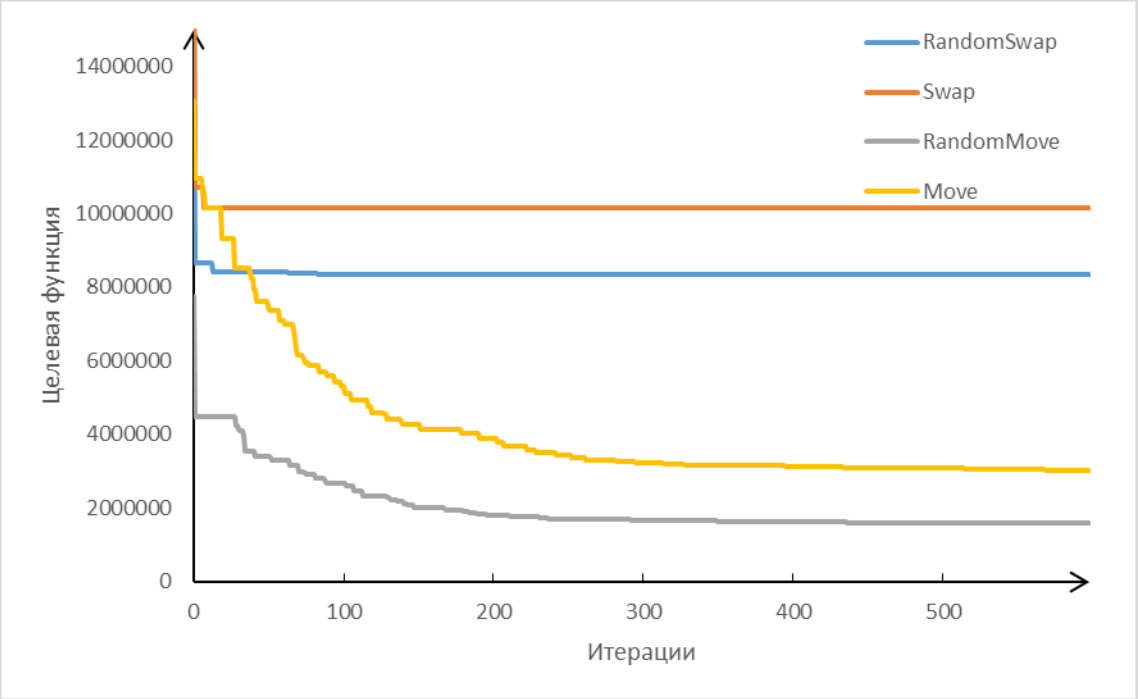



Рисунок 1. Сравнение просмотра всей окрестности и рандомизированной окрестности

Если учесть, что операцию Swap можно представить в виде двух операций Move, то можно предложить не использовать окрестность для Swap. Но это ошибочное предложение. С точки зрения метода локального поиска Swap не эквивалентен Move, потому что сделав один Move, второй шаг, дополняющий до выбранного Swap, может быть не выбран алгоритмом. И более того использование объединения рандомизированных окрестностей Move и Swap дает заметные улучшения результата (таблица 1).

Объединение рандомизированных окрестностей Move и Swap

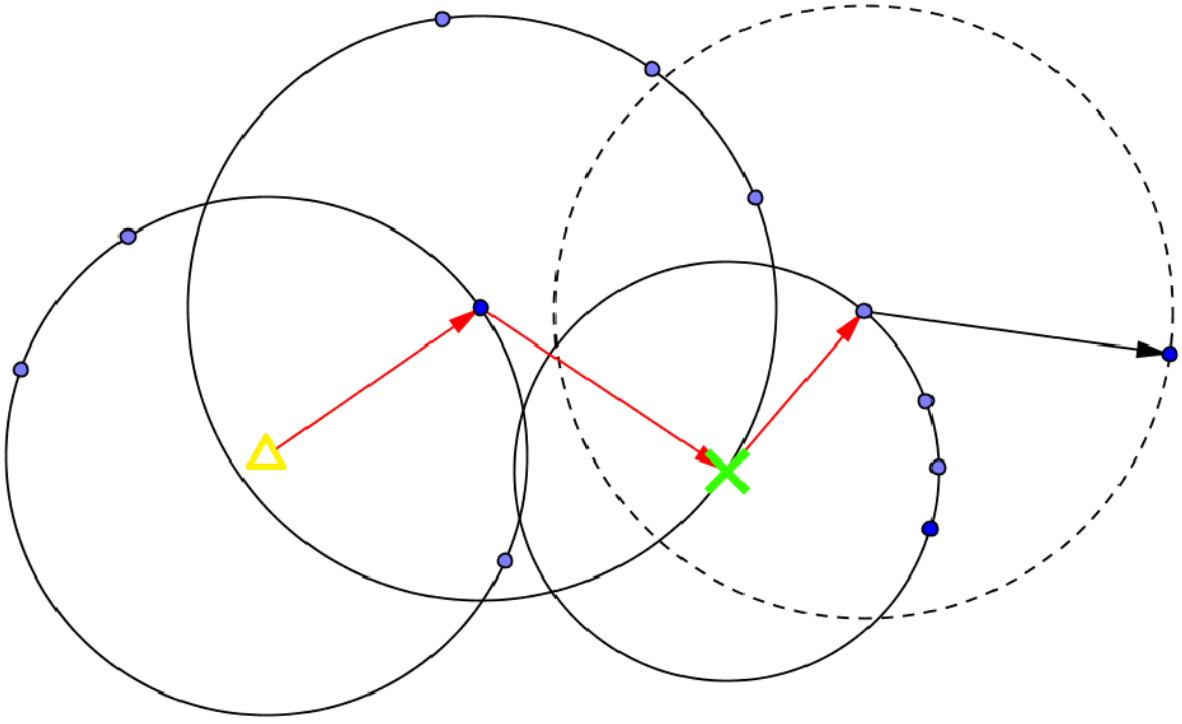
1. Выполнить шаги 1 и 2 из описания «Рандомизированная окрестность к Move» и «Рандомизированная окрестность к Swap»
2. Применить к текущему решению операцию, дающую лучшее решение.

Окрестность *Assign*

С каждого сервера нужно изъять по одному случайному диску. Затем составить матрицу размера $|S|^2$, где в клетку $[i][j]$ запишем значение перегрузки j-го сервера при вставке i-го выбранного диска. Решить задачу о назначениях, т.е. каждому серверу назначить диск так, чтобы перегрузка была минимальной. Для задачи о назначениях есть точный полиномиальный алгоритм – Венгерский алгоритм $O(n^3)$ [16]

Окрестность Lin-Kernighan

Эта окрестность успешно применяется для задачи коммивояжера [15]. Но идею можно применять и в других задачах. Окрестность будем строить в соответствии с рисунком 2: начинаем из текущего решения. Используя некоторое рандомизированное множество соседних решений по окрестностям Move или Swap, выберем соседа с лучшей целевой функцией среди этих соседей (возможно этот лучший сосед будет иметь целевую функцию хуже, чем начальное решение). Продолжим поиск с лучшего соседа. Повторяем процедуру $N$ раз, если было найдено решение, которое лучше стартового, заменяем стартовое решение на лучшее.



*На рисунке желтым треугольником обозначено начальное решение, а зеленым крестом новое решение. Синие круги – решения, просматриваемые при переходах по окрестностям Move и Swap.*

Рисунок 2. Схематическое изображение перехода по окрестности Lin-Kernighan

Локальный поиск очень прост в реализации и быстро дает довольно хорошие результаты. Одним из главных недостатков данного метода является сходимость к

локальному оптимуму. Более того алгоритм локального поиска может сильно зависеть от начального решения. Существует много алгоритмов, позволяющих не застревать в локальных оптимумов[14]: GRASP, VNS, multistart, имитация отжига, поиск с запретами и т.д. [17, 18, 20] Генетические алгоритмы справляются с этой задачей путем итеративного применения локального поиска к каждому новому поколению, которые получаются применением операторов скрещивания и мутации.

Критерий остановки

В качестве критерия можно выбрать заданное количество итераций или остановку по времени. Вычислительный эксперимент (таблица 1) показал, что значительное улучшение решения можно получать за 10 – 20 минут. Этому времени для разных окрестностей соответствует разное количество итераций в диапазоне от 1000 до 2000.

### 2.2.3 Оператор скрещивания

На вход оператору скрещивания подается несколько родительских решений, чаще всего два, а на выходе получается множество решений потомков. Эти операторы помогают диверсифицировать популяцию решений и выбираться из локальных оптимумов. Более того, если удается выделить хорошие свойства решений и организовать скрещивание так, что эти свойства наследуются потомками, то возможно получать улучшения родительских решений. Рассмотрим оператор скрещивания (рис.3), основанный на построении пути между решениями[20]. Между двумя данными решениями, применяя операторы Move и Swap, можно построить цепочку решений – путь. На этом пути, возможно, найдутся решения с лучшей целевой функцией. На самом деле, между двумя решениями найдется множество путей, потому что если у нас есть один путь (последовательность Move и Swap), то всевозможные перестановки этих операторов дадут новые пути. Будем выбирать путь по следующему правилу: к решению с меньшим значением перегрузки будем применять оператор Move, приближающий к решению с большей перегрузкой. Диски, к которым будем применять оператор Move, выбираем следующим образом:

- Записать в вектор все диски, расположения которых отличаются в родительских решениях
- Перемешать вектор
- Применять оператор Move к дискам по порядку следования в векторе, пока удовлетворяются ограничения на ресурсы серверов

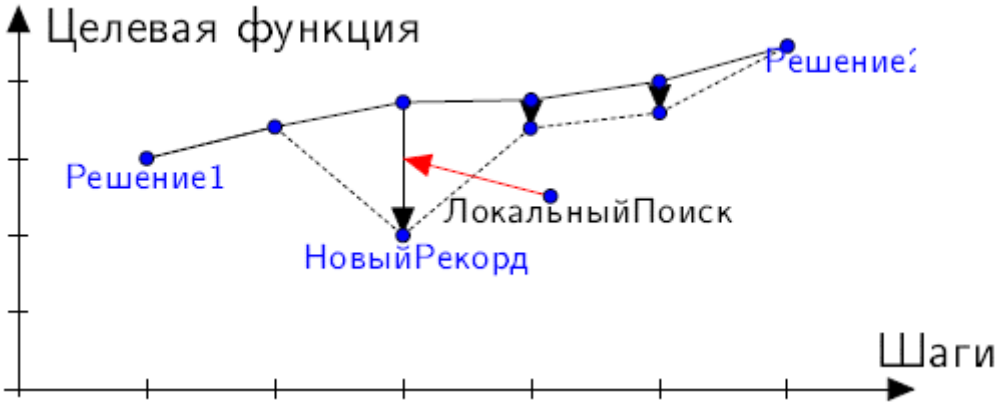



Рисунок 3. Алгоритм скрещивания PathRelinking

Применяя к каждому промежуточному решению алгоритм локального поиска, возможно получать лучшие решения. Результат скрещивания - несколько лучших решений из составленного пути.

### 2.2.4 Оператор мутации

Оператор мутации - это унарный оператор, действующий на единственное решение. Мутации представляют собой незначительные случайные изменения выбранного представителя популяции. Для оператора мутации важно сохранить допустимость решения, т.е. не нарушить накладные расходы на «изъятие» и «вставку» дисков.

В силу того, что локальный поиск проводится по рандомизированным окрестностям и алгоритм генерации начальной популяции задает достаточную диверсификацию популяции, чтобы на протяжении всей работы алгоритма решения в популяции значительно различались, то было принято решение не применять оператор мутации.

# 3 Численные эксперименты

## 3.1 Описание тестовых данных

Тестовые данные были предоставлены компанией, занимающейся облачным хостингом.

Первый класс примеров $|D|=200, |S|=20, |R|=6, \mathrm{T}=1008$, т.е. 200 дисков размещаются на 20 серверах и для каждой из 6 характеристик проводится 1008 замеров нагрузки (или 1 замер каждые 15 минут в течение недели). Начальное решение имеет существенную перегрузку, но имеются почти пустые сервера. При использовании прогнозов нагрузки метки классов назначались следующим образом: $|D_0|=3$, $|D_1|=0,7,15$, т.е. выбиралось 3 диска, которые считались большими и 0, 7 или 15 дисков – потенциально большими, соответственно.

Второй класс примеров $|D|=2680, |S|=20, |R|=6, \mathrm{T}=1008$, т.е. были предоставлены реальные данные для большего количества дисков. Этот пример достаточно простой и удается найти оптимум, т.е. избавиться от перегрузки. Примеры с $|D|=10720,\ 26800$ генерировались дублированием и перестановками дисков из примера с $|D|=2680$.

## 3.2 Выбор размера начальной популяции

Для выбора размера начальной популяции на первом классе примеров был запущен генетический алгоритм с локальным поиском по объединению окрестностей Move и Swap, с критерием остановки по времени равным 10 минутам (значительное время для такого небольшого примера).


Рисунок 4. Выбор размера начальной популяции

В соответствии с риуснком 4 можно сделать вывод, что хорошие результаты получаются с достаточно небольшим размером популяции. Объясняется это тем, что выбранная процедура генерации начальной популяции достаточно затратная по вычислительным ресурсам и время с большей пользой можно потратить на выполнение локального поиска и применение оператора скрещивания. Но за большее время, вполне вероятно, что чем крупнее начальная популяция, тем больше она сможет покрыть пространство решений и найти лучшее решение.

### 3.3 Выбор параметров для окрестности Lin—Kernighan

Тестовые данные: пример из первого класса без учета классов дисков. Время работы 5 минут. На рисунках 5 и 6 представлены усредненные результаты работы алгоритма за 10 запусков.

Для окрестности Lin-Kernighan, описанной в 3.2.2, исследовалось поведение алгоритма при изменении числа N – количества шагов по объединению рандомизированных окрестностей Move и Swap. На рисунке 5 представлены результаты работы алгоритма при изменении параметра N. Хорошие результаты дает увеличение количества шагов до 90 – 100, дальнейшее увеличение дает незначительное уменьшение целевой функции.


Рисунок 5. Количество шагов Move или Swap в окрестности Lin-Kernighan

Второй параметр определяет размер рандомизированных окрестностей Move и Swap, т.е. сколько случайных ходов Move и Swap следует рассматривать на каждой

итерации. Размеры окрестностей Move $|D||S| = 200 * 20 = 4000$ и Swap $|D|^2 = 40000$. С учетом допустимости ходов эти окрестности становятся гораздо меньше. В данном случае можно сделать алгоритм линейно зависимым от размера входных данных, ограничив размер рандомизированной окрестности величиной $C*|D|$, где $C$ - это константа, а $|D|$ - количество дисков в задаче. Выбор параметров производился с помощью вычислительного эксперимента на примером с $|D| = 200$, поэтому константу $C$ в соответствии с рисунком 6 можно выбирать из интервала $[0,1;0,5]$. Увеличение значения $C$ дает незначительное улучшение целевой функции.

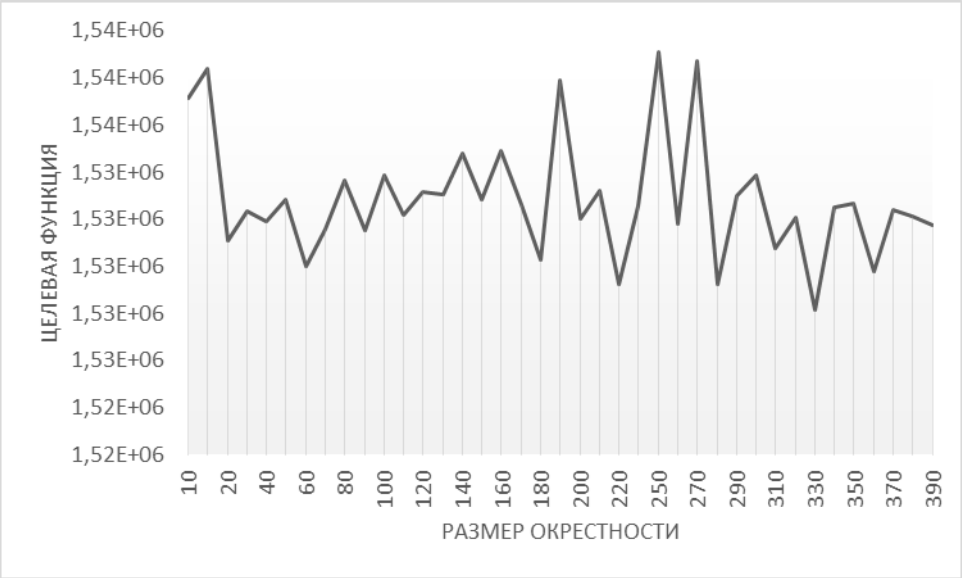



Рисунок 6. Зависимость целевой функции от размера рандомизированных окрестностей

### 3.4 Сравнение окрестностей

Сравнение окрестностей производилось на тестовых данных первого класса с начальной перегрузкой $F_{GA}^{0} = 5.60706*10^{7}$

Таблица 1. Сравнение окрестностей на примере без классов

| Итерации | Окрестность | $t_{GA}$, сек | $F_{GA}, 10^{6}$ |
|---|---|---|---|
| 1078 | Move | 600 | 1.61933 |
| 1552 | Swap | | 6.72452 |
| 1178 | Move + Swap | | 1.53738 |
| 675 | LinKernighan | | 1.53275 |
| 1832 | Move | 1200 | 1.55846 |
| 3024 | Swap | | 6.09213 |
| 2337 | Move + Swap | | 1.53709 |
| 1249 | LinKernighan | | 1.52946 |

Из таблицы 1 можно сделать вывод, что объединение окрестностей Move и Swap дает сравнимые Lin-Kernighan, основанном на объединение окрестностей Move и Swap, но в среднем немного хуже.

Хотя операция Swap является более сложной, чем Move алгоритм успевает выполнить больше итераций, потому что Swap тратит больше ресурсов на перемещение дисков и многие ходы становятся запрещены.

При проведении экспериментов с заданными прогнозами по нагрузке (т.е. классами дисков) на первом классе тестовых данных оказалось (таблица 2), что вариация количества потенциально больших дисков в диапазоне от нуля до количества серверов, оставшихся после выделения серверов с большими дисками, не существенно влияет на значение целевой функции

Таблица 2. Работа алгоритма при различных прогнозах нагрузки

| Итерации | Больших дисков | Потенциально больших дисков | $t_{GA}$, сек | $F_{GA}, 10^6$ |
|---|---|---|---|---|
| 500 | 3 | 0 | 600 | 1.60994 |
| | | 5 | | 1.62042 |
| | | 15 | | 1.61953 |

**3.5 Испытания алгоритма при значительном увеличении размерности задачи**

На втором классе тестовых данных можно исследовать поведения алгоритма на задачах большой размерности.

Из-за выбора представления решения алгоритм имеет линейную сложность по памяти. По результатам можно судить, что алгоритм способен улучшать решение за приемлемое время. В примерах с $|D| = 10720, 26800$, сервера испытывают сильную перегрузку и тяжело получить значительные улучшения, но даже такое небольшое улучшение может принести пользу компании.

Таблица 3. Испытание алгоритма на задачах большой размерности

| $\|D\|$ | $t_{GA}$, *мин* | RAM, MB | $F_{GA}^0$ | $F_{GA}$ |
|---|---|---|---|---|
| 2680 | 10 | <300 | 72038 | 0 |
| | 20 | | | 0 |
| 10720 | 10 | <700 | $2.57277 * 10^7$ | $2.34547 * 10^7$ |
| | 20 | | | $2.33259 * 10^7$ |
| 26800 | 10 | <1500 | $1.30869 * 10^8$ | $1.30354 * 10^8$ |
| | 20 | | | $1.30243 * 10^8$ |

### 3.6 Сравнение с другими метаэвристиками

Применение окрестности Lin-Kernighan в генетическом алгоритме позволило значительно улучшить результаты по сравнению с поиском с запретами [1].

Таблица 4. Сравнение генетического алгоритма и поиска с запретами

| Окрестность | Большие диски | Потенциально большие диски | $t_{TS}$ , сек | $F_{TS}$ | $t_{GA}$ , сек | $F_{GA}$ |
|---|---|---|---|---|---|---|
| Move | 3 | 15 | 673 | $5.56*10^6$ | 600 | $1.61*10^6$ |
| | 0 | 0 | 690 | $3.76*10^6$ | | $1.53*10^6$ |
| Swap | 3 | 15 | 3580 | $6.10*10^6$ | | $1.61*10^6$ |
| | 0 | 0 | 3480 | $4.50*10^6$ | | $1.53*10^6$ |
| Объединение Move и Swap | 3 | 15 | 346 | $7.7*10^6$ | | $1.61*10^6$ |
| | 0 | 0 | 483 | $6.4*10^6$ | | $1.53*10^6$ |

**Заключение**

В данной работе проводилось исследование применения генетического алгоритма для достаточно новой задачи балансировки нагрузки на серверы. Решение задачи в целочисленной постановке с помощью известных решателей не приводят к удовлетворительным результатам.

В работе получены следующие результаты:

- Разработан генетический алгоритм для решения задачи балансировки нагрузки на серверы. Алгоритм реализован на C++
- Исследованы различные окрестности для реализации метода локального поиска, проведено сравнение на основе вычислительного эксперимента
- Проведено исследование поведения алгоритма при различных входных данных, в том числе при значительном увеличении размерности задачи.
- Даны рекомендации по выбору параметров алгоритма

Алгоритм для задачи в постановке (1-4) представлен на X Международной Азиатской школе-семинаре «Проблемы оптимизации сложных систем». [8]

Улучшенный алгоритм и решение задачи с различными классами дисков с использованием реализованного алгоритма были представлены на Международной Научной Студенческой Конференции (МНСК-2015) в секции «Теоретическая кибернетика», работа «Генетический алгоритм для балансировки нагрузки на серверы». [7]

**Литература**


1. Бежецков, Д.Е. Поиск с запретами для задачи балансировки нагрузки на серверы // Материалы 53-й Международной научной студенческой конференции МНСК-2015: Математика. – Новосибирск, 2015. – С. 101.

2. Бежецков, Д.Е. Поиск с запретами для задачи балансировки нагрузки на серверы // Труды X международной Азиатской школы-семинара «Проблемы оптимизации сложных систем». Алматы: НЦ НТИ. – 2014. – Часть 1. – С. 125-130.

3. Глебов, Н.И. Методы оптимизации. Учебное пособие. / Ю.А. Кочетов, А.В. Плясунов // Новосибирск: Новосибирский государственный университет, 2000. - 105 с.

4. Давыдов, И.А. Локальный поиск с запретами для дискретной задачи о $(r|p)$-центроиде // Дискрет. анализ и исслед. операций. Новосибирск: Изд. ИМ, 2012. - Том 19. - № 2. - С. 19–40.

5. Давыдов, И.А. Локальный поиск с экспоненциальной окрестностью для задачи балансировки нагрузки на серверы / П.А. Кононова, Ю.А. Кочетов // Дискретный анализ и исследование операций. Новосибирск: Изд. ИМ, 2014. – Том 21. - №6. – С. 21–34.

6. Давыдов, И.А. Быстрые метаэвристики для дискретной задачи о $(r|p)$ −центроиде / Ю.А. Кочетов, Н. Младенович, Д. Уросевич // Автоматика и телемеханика. 2014. - № 4. - С. 106–119.

7. Дорожко, А.В. Генетический алгоритм для задачи балансировки нагрузки на серверы // Материалы 53-й Международной научной студенческой конференции МНСК-2015: Математика. – Новосибирск, 2015 – С. 112.

8. Дорожко, А.В. Генетический алгоритм для задачи балансировки нагрузки на серверы // Труды X международной Азиатской школы-семинара «Проблемы оптимизации сложных систем». Алматы: НЦ НТИ. – 2014. – Часть 1. – С. 254-260.

9. Иванов, С. А. Алгоритм имитации отжига для задачи балансировки нагрузки на серверы // Материалы 53-й Международной научной студенческой конференции МНСК-2015: Математика. – Новосибирск, 2015 – С. 115.

10. Кочетов, Ю.А. Методы локального поиска для дискретных задач размещения Модели и алгоритмы. Saarbrucken: Lambert Academic Publishing. - 2011. - 259 с.
11. Кочетов, Ю.А. Задача балансировки нагрузки на серверы / Н.А. Кочетова // Вестник НГУ. Серия Информационные технологии. Новосибирск, 2013. - Том 11. - № 4. - С. 71-76.
12. Кочетов, Ю.А. Локальный поиск с чередующимися окрестностями / Н. Младенович, П. Хансен // Дискретный анализ и исследование операций. Новосибирск: Изд. ИМ, 2003. - Том 10. - № 1. - С.11–43.
13. Мельников, А.А. Рандомизированный локальный поиск для дискретной задачи конкурентного размещения предприятий // Автоматика и телемеханика. 2014. - № 4. - С. 134–152.
14. Пападимитриу, Х. Комбинаторная оптимизация. Алгоритмы и сложность. / К. Стайглиц. - М. : Мир, 1985. – 512 с.
15. Aarts, E. Local Search in Combinatorial Optimization. / J.K. Lenstra. – John Wiley & Sons, Inc. New York, NY, USA, 1997. - p.216-286.
16. Burkard, R.E. Assignment Problems. / M. Dell'Amico, S. Martello. - SIAM: Philadelphia (PA.), 2009. – 415 p.
17. Dreo, J. Metaheuristics for Hard Optimization. / A. Petrowski, P. Siarry, E. Taillard. – Springer, 2006. – 369 p.
18. Glover, F. Tabu Search./ M. Laguna. – Boston : Kluwer Academic Publishers, 1997. – 382p.
19. Holland, J. Adaptation in natural and artificial systems. – University of Michigan Press, 1975. – 211p.
20. Talbi, El. Metaheuristics : from design to implementation. — New Jersey : John Wiley & Sons, 2009. – 593p.