# ГЛАВА 2. РАЗРАБОТКА МАТЕМАТИЧЕСКОЙ МОДЕЛИ АВТОМАТИЧЕСКОГО РАЗВЕРТЫВАНИЯ МИКРОСЕРВИСНЫХ ПРИЛОЖЕНИЙ В ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ

## 2.1 Разработка математической модели системы

Проблема задействования наименьшего количества серверов при распределении на них программных компонентов, описанная в главе 1, более подробно звучит следующим образом: «имеется коллекция программных компонентов (далее будем называть их сервисами) и коллекция виртуальных машин (далее будем называть их серверами). Необходимо распределить все сервисы на минимальное количество серверов в предположении, что один сервис может располагаться только на одном сервере». Данная задача может быть описана с использованием терминологии комбинаторной оптимизации, как упаковка множества на , где упаковка — это подсемейство  множеств, такое, что все множества из  попарно не пересекаются, т.е. набор серверов, в которых сервис не входит в два разных сервера, – множество серверов, а - семейство подмножеств серверов. Эта задача, в свою очередь, является классической NP-полной задачей в теории вычислительной сложности и комбинаторике и более известна, как задача об упаковке в контейнеры [19, 20].

Неотъемлемой частью алгоритма оптимизации является критерий оптимальности – количественная мера оптимизируемого качества объекта. Значение критерия оптимальности выражается целевой функцией. Таким образом, задача оптимизации сводится к нахождению минимума или максимума целевой функции.

Приведем постановку задачи размещения сервисов на ограниченном количестве серверов. Возьмем - множество сервисов и - множество серверов. Рассмотрим сервисы и сервера с точки зрения задачи упаковки, как предметы и контейнеры, тогда – размер -го предмета (сервиса), а – вместимость го контейнера (сервера). Тогда, требуется найти такое разбиение множества на непересекающиеся подмножества , чтобы сумма размеров сервисов в каждом подмножестве не превосходила заданную вместимость конкретного сервера, и чтобы было наименьшим возможным.

Математическая формулировка данной задачи следующая:

*,*

*,*

где

Можно считать, что сервисы, принадлежащие каждому множеству , размещаются на сервера разного размера, а цель состоит в размещении сервисов из множества на как можно меньшее число серверов .

Критерием оптимальности в данном случае будем считать число серверов . Решение считается тем более оптимальным, чем ниже значение .

Каждый сервис описывается кортежем [15] , где - количество занимаемой оперативной памяти, - количество занимаемой памяти на постоянном запоминающем устройстве, - количество занимаемого процессорного времени, – тип операционной системы, на которой запускается -й сервис. Введем следующие ограничения:

где и – верхние пределы размера файла сервиса, выражаемые в гигабайтах, – верхний предел занимаемого процессорного времени, выраженный в условных единицах, а – элементы множества типов операционных систем.

Каждый сервер описывается кортежем , где - количество свободной оперативной памяти, - общий объем оперативной памяти, - количество свободной памяти на постоянном запоминающем устройстве, – общий объем памяти на постоянном запоминающем устройстве, – общее количество процессорного времени, – тип операционной системы сервера. Введем следующие ограничения:

где и – верхние границы размера файла сервиса, выражаемые в гигабайтах, – верхняя граница занимаемого процессорного времени, выраженный в условных единицах, а – элементы множества типов операционных систем.

В ходе решения задачи положение элементов, обозначенное заданным типом, в пространстве задается множеством .

## 2.2 Описание существующих алгоритмов комбинаторной оптимизации

Для NP-полных вычислительных задач не существует строгого алгоритма решения. (не сущ. Алгоритма решения за полиномиальное время) Трудоёмкость таких задач экспоненциально растёт с увеличением объема данных. Алгоритмы с экспоненциальным временем считаются неэффективными, поэтому рассмотрим ряд эвристических алгоритмов, позволяющих найти решение, находящееся близко к оптимальному по такому-то критерию.

Простейшими [алгоритмами](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) упаковки являются [2]:

* 1. Алгоритм «Следующий подходящий» (Next fit, NF). Упаковка предметов происходит в произвольном порядке по следующему правилу: первый предмет помещается в первый контейнер. На k-м шаге помещается k-й предмет в текущий контейнер. Если предмет помещается, то переходим к следующему шагу, иначе помещаем предмет в новый контейнер.
  2. Алгоритм «Первый подходящий» (First fit, FF). Первый предмет помещается в первый контейнер. На k-м шаге ищется контейнер с наименьшим номером, куда помещается k-й предмет, и помещаем его туда. Если такого контейнера нет, то берем новый пустой контейнер и помещаем предмет в него.
  3. Алгоритм «Наилучший подходящий» (BF). В произвольном порядке упаковываем предметы по следующему правилу: первый предмет помещаем в первый контейнер. На k-м шаге размещаем k-й предмет. Находим частично заполненные контейнеры, где достаточно для него свободного места и выбираем среди них наиболее заполненный. Если таких нет, то берем новый пустой контейнер и помещаем k-й предмет в него.
  4. Алгоритм «Первый подходящий с упорядочиванием» (First fit decreasing, FFD).

Предметы сортируются по невозрастанию весов w1 ≥ w2 ≥ … ≥ wn и применяется алгоритм FF.

* 1. Алгоритм «Наилучший подходящий с упорядочиванием» (Best fit decreasing, BFD). Предметы сортируются по невозрастанию весов w1 ≥ w2 ≥ … ≥ wn и применяется алгоритм BF.

По каждому из вышеупомянутых алгоритмов нужна блок-схема?

Алгоритмы NF, FF, BF являются on-line алгоритмами. При работе таких алгоритмов упакованный предмет нельзя перемещать в другой контейнер, например, упаковка на конвейере. Алгоритмы FFD и BFD используются тогда, когда существует возможность доступа ко всем контейнерам [1].

Идеи локального поиска получили свое дальнейшее развитие в так называемых метаэвристиках, то есть в общих схемах построения алгоритмов, которые могут быть применены практически к любой задаче дискретной оптимизации. Идея этих методов основана на предположении, что целевая функция имеет много локальных экстремумов, а просмотр всех допустимых решений невозможен, несмотря на конечность их числа. В такой ситуации нужно сосредоточить поиск в наиболее перспективных частях допустимой области. Таким образом, задача сводится к выявлению таких областей и быстрому их просмотру. Каждая из метаэвристик решает эту проблему по‐своему [3]. Рассмотрим генетический алгоритм.

Генетический алгоритм – это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе. Цель генетического алгоритма при решении задачи оптимизации состоит в том, чтобы найти приближённое решение, близкое, но не гарантированно оптимальное решение. Впервые эти нестандартные идеи были применены к решению оптимизационных задач в середине 70-х годов [4]. Примерно через десять лет появились первые теоретические обоснования этого подхода [5,6]. На сегодняшний день генетические алгоритмы доказали свою конкурентоспособность при решении многих NP-трудных задач [7,8,9]. Применение генетических методов для решения NP-трудных комбинаторных задач оптимизации полезно тогда, когда необходимый объем вычислительных затрат может оказаться большим, но скорость, с которой этот объем увеличивается при экспоненциальном росте размерности задачи дискретной оптимизации, часто может расти лишь линейно [10].

В теории генетических алгоритмов применяется следующая терминология [11]:

· ген (свойство) – атомарный элемент хромосомы. Ген может быть битом, числом или неким другим объектом;

· аллель – значение конкретного гена;

· локус – положение конкретного гена в хромосоме;

· хромосома (цепочка) – упорядоченная последовательность генов;

· генотип (код) – упорядоченная последовательность хромосом;

· особь (индивидуум) – конкретный экземпляр генотипа;

· фенотип – аргумент (набор аргументов) целевой функции, соответствующий генотипу (т.е. интерпретация генотипа с точки зрения решаемой задачи).

Структура популяции представлена на рис. 2.1.

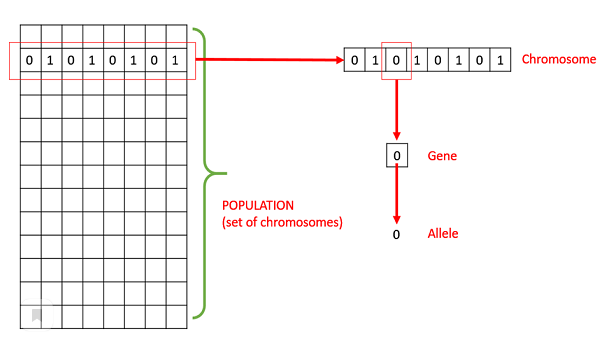


Рисунок 2.1 – Структура популяции

Поиск решения задачи с помощью генетического алгоритма заключается в использовании итерационной процедуры, когда исходное решение с каждым шагом постепенно улучшается. В отличие от методов математического программирования на каждой итерации рассматриваются сразу несколько альтернативных вариантов (особей) решения задачи. Совокупность особей, используемых в итерации, называется популяцией. Количество особей в популяции называют размером популяции.

На каждой итерации генетический алгоритм обновляет популяцию путем создания новых особей и уничтожения худших. Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием [функции приспособленности](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8), в результате чего с каждым генотипом ассоциируется определённое значение - приспособленность, которое определяет насколько хорошо [фенотип](https://ru.wikipedia.org/wiki/%D0%A4%D0%B5%D0%BD%D0%BE%D1%82%D0%B8%D0%BF), им описываемый, решает поставленную задачу. Функция приспособленности - вещественная или целочисленная [функция](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) одной или нескольких переменных, подлежащая [оптимизации](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) в результате работы [генетического алгоритма](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), направляет эволюцию в сторону оптимального решения. К функции приспособленности применяются следующие требования:

1. Функция должна быть адекватно заданной. Это означает, что распределение значений должно совпадать с распределением реального качества решений.
2. Функция должна иметь разнообразный рельеф, без больших «плоских» участков, так как иначе, несмотря на то что решения будут различаться, они будут иметь одинаковую оценку, а значит при работе алгоритма не будет возможности выбрать лучшее решение, направление дальнейшего развития. Эта проблема еще упоминается как «[проблема поля для гольфа](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%B0_%D0%BF%D0%BE%D0%BB%D1%8F_%D0%B4%D0%BB%D1%8F_%D0%B3%D0%BE%D0%BB%D1%8C%D1%84%D0%B0&action=edit&redlink=1)», где все пространство абсолютно одинаково, за исключением лишь одной точки, и является оптимальным решением - в этом случае алгоритм просто остановится или поиск решения будет происходить совершенно случайным образом.
3. Функция приспособленности должна требовать минимум ресурсов. Поскольку это наиболее часто используемая деталь алгоритма, она оказывает существенное влияние на его скорость работы [18].

Генерация новых особей происходит на основе моделирования процесса размножения с помощью оператора скрещивания. Порождающие особи называются родителями, а порожденные - потомками. Выбор родителей (пары кодовых строк) для скрещивания выполняется различными способами, наиболее известные из которых метод колеса рулетки, когда хромосома выбирается случайным образом, а вероятность ее выбора зависит от ее приспособленности [11], селекция отсечением, когда гены выбираются на основе значения функции приспособленности, превышающей заданный порог, турнирная селекция, когда из популяции, содержащей хромосом, выбирается случайным образом хромосом (тур), и лучшая хромосома из тура попадает в родительскую популяцию. Родительская пара, как правило, порождает пару потомков. Непосредственная генерация новых кодовых строк из двух выбранных происходит за счет применения операции кроссовера, которая также, как и селекция, имеет разные варианты, такие как бинарный и всеобщий кроссовер. Бинарный кроссовер делится на одноточечное многоточечное скрещивание. Одноточечное скрещивание выполняется следующим образом:

– на первом шаге происходит случайный выбор пары хромосом;

– на втором шаге случайно выбирается n – точка скрещивания. Если генами являются отдельные биты, то n является номером бита по длине хромосомы, иначе n – номер гена;

– на третьем шаге происходит сама операция, которая заключается в переписывании генов одной хромосомы в другую хромосому и наоборот, начиная с точки скрещивания. Таким образом, одноточечное скрещивание рассматривается просто как обмен частями хромосом, начиная с некоторой случайной позиции – точки скрещивания.

Многоточечное скрещивание предполагает выбор нескольких несовпадающих точек скрещивания. Хромосомы обмениваются частями после каждой точки скрещивания.

При всеобщем кроссовере каждая позиция по длине хромосомы может быть потенциальной точкой скрещивания. Здесь также случайным образом выбираются две родительские хромосомы, затем для каждого потомка генерируется случайным образом бинарный вектор маски, длина которого совпадает с длиной хромосомы. Нулевое значение бита вектора маски указывает на то, что значение бита потомка должно быть взято из первой родительской хромосомы, единичное значение – из второй родительской хромосомы.

Изменение особей осуществляется за счет работы оператора мутации, применяемого к случайно выбранным потомкам за счет изменения случайного выбранного гена (генов). Оператор мутации используется для внесения случайных изменений в хромосомы особей. Это позволяет алгоритму выходить из локальных экстремумов и, тем самым, эффективнее исследовать пространство поиска. Поскольку размер популяции фиксирован, то порождение потомков должно сопровождаться уничтожением особей. Выбор лучших («жизнеспособных») особей из числа родителей и потомков выполняется в операторе редукции, который уничтожает худшие («малоприспособленные») особи. Основным правилом отбора является закон эволюции: «выживает сильнейший», который обеспечивает улучшение искомого решения.

При недостаточном значении вероятности мутации и попадания алгоритма в локальный экстремум становится возможной преждевременная сходимость генетического алгоритма (также говорят о вырождении популяции), когда решение еще не найдено, но в популяции практически все особи становятся одинаковыми и долгое время (порядка нескольких десятков и сотен поколений) не наблюдается улучшения приспособленности. Преждевременная сходимость популяции может быть исправлена путем:

* изменения стратегии выбора родительских пар для скрещивания;
* отслеживания появления в популяции идентичных особей и их удаления;
* использования сильно разрушающего оператора кроссовера;
* увеличения вероятности мутации.

Операторы скрещивания, мутации и редукции называют генетическими операторами. Скрещивание и мутация выполняются с использованием элементов случайности, а редукция – по строго определенным (детерминированным) правилам.

Критерием остановки работы генетического алгоритма может быть одно из следующих событий:

* сформировано заданное число поколений;
* исчерпано время, отведенное на эволюцию;
* популяция достигла заданного качества (значение критерия одной (нескольких, всех) особей превысило заданный порог);
* достигнут некоторый уровень сходимости (особи в популяции стали настолько подобными, что дальнейшее их улучшение происходит чрезвычайно медленно);

Блок-схема генетического алгоритма представлена на рисунке 2.2.



Рисунок 2.2. Блок-схема генетического алгоритма

Наряду с генетическим алгоритмом для решения многих оптимизационных задач, в т.ч. комбинаторных [13, 14], применяется метод отжига, также известный, как алгоритм имитации отжига – алгоритм оптимизации, использующий упорядоченный случайный поиск на основе аналогии с процессом образования в веществе кристаллической структуры с минимальной энергией при охлаждении.

Метод отжига служит для поиска глобального минимума некоторой функции , заданной для x из некоторого пространства , дискретного или непрерывного. Элементы множества представляют собой состояния воображаемой физической системы (энергетические уровни), а значение функции в этих точках используется, как энергия системы . В каждый момент предполагается заданной температура системы , уменьшающаяся с течением времени. После попадания в состояние при температуре , следующее состояние системы выбирается в соответствии с порождающим семейством вероятностных распределений , которое при фиксированных и задает случайный элемент со значениями в пространстве . После генерации нового состояния система с вероятностью переходит к следующему шагу в состояние , в противном случае, процесс генерации повторяется. Здесь означает приращение функции энергии . Величина называется вероятностью принятия нового состояния. Как правило, в качестве функции выбирается либо точное значение соответствующей физической величины , либо приближенное значение . Вторая формула используется наиболее часто. При ее использовании оказывается больше единицы в случае , и тогда соответствующая вероятность считается равной 1. Таким образом, если новое состояние дает лучшее значение оптимизируемой функции, то переход в это состояние произойдет в любом случае. Существуют различные законы уменьшения температуры с течением времени. Для Больцмановского отжига [16] используется закон . Доказано [16], что для достаточно больших и общем количестве шагов , выбор такого закона гарантирует нахождение глобального минимума, однако, скорость работы алгоритма относительно невысока, например, чтобы понизить исходную температуру в 40 раз, требуется итераций. При использовании сверхбыстрого отжига по закону , где D – размерность пространства состояний, для решения задач в пространстве размерности больше 1, скорость алгоритма меньше, чем при применении Больцмановского закона. Также, существует метод под названием «Сверхбыстрый отжиг», в этом случае температура изменяется по закону , , где – декремент затухания температуры.

Вероятность пребывания системы в состоянии с энергией при температуре равна , где С — нормировочный множитель. Это свойство общее для любых макроскопических систем и известно под названием распределения Гиббса [17].

Ниже приведена блок-схема алгоритма имитации отжига (рис. 2.3).



Рисунок 2.3. Блок схема алгоритма имитации отжига.

## 2.3 Разработка симулятора сетевой инфраструктуры

При проектировании системы оптимизации автоматического развертывания микросервисных приложений в облачной инфраструктуре, необходимо создать симулятор данной системы, состоящий из программного обеспечения для хранения и получения данных об облачной инфраструктуре и программного обеспечения, целью которого является решение задачи распределения сервисов на сервера. Симулятор - система, которая ведет себя подобно настоящей системе, но реализована совершенно по-другому. Симулятор обеспечивает базовое поведение системы, но может не обязательно соответствовать всем правилам моделируемой системы. Симулятор облачной инфраструктуры необходим для моделирования работы системы распределения микросервисов в данной инфраструктуре. Программная реализация алгоритмов распределения сервисов на сервера будет описана в следующих пунктах.

Критериями выбора технологии реализации симулятора облачной инфраструктуры являются:

1. Возможность работы приложения в локальной вычислительной сети.
2. Использование в приложении для обмена данными открытых и стандартизованных форматов хранения и обмена информацией.
3. Кроссплатформенность

Исходя из вышеперечисленных требований была выбрана модульная платформа для разработки программного обеспечения с открытым исходным кодом .NET. Симулятор облачной инфраструктуры – это веб-приложение, созданное на платформе .NET с использованием языка программирования C#. Для хранения данных используется система управления базами данных класса NoSQL с открытым исходным кодом Redis, работающая со структурами данных типа «ключ — значение». Входные и выходные данных представлен в формате JSON в приложении А. Для записи и получения данных используется веб-api. При запросе данных об облачной инфраструктуре приложение возвращает информацию о каждом сервере и развернутом на нем программного обеспечении. Эти данные являются входными при работе алгоритма хостирования сервисов на сервера. Также, в процессе анализа работы вышеописанных алгоритмов потребуется вносить изменения в симулируемую инфраструктуру, для этого симулятор имеет API для записи данных. Формат входных данных - JSON. На рисунках 2.5 и 2.6 представлены блок-схемы алгоритмов запроса и записи данных в симулятор инфраструктуры. Структурная схема симулятора показана на рисунке 2.4.



Рисунок 2.4. Структурная схема симулятора облачной инфраструктуры



Рисунок 2.5. Блок-схема алгоритма запроса данных у симулятора инфраструктуры



Рисунок 2.6. Блок-схема алгоритма записи данных в базу данных симулятора инфраструктуры

## 2.4 Реализация алгоритма «наилучший подходящий с упорядочиванием»

Для решения задачи распределения сервисов по серверам с помощью эвристических алгоритмов будем использовать offline-алгоритм «BFD» или «Наилучший подходящий с упорядочиванием». Выбор данного алгоритма обусловлен тем, что в процессе упаковки производится поиск наиболее заполненных серверов, куда может быть размещен каждый сервис, т.е. при использовании данного алгоритма мы будем стремиться использовать наименьшее количество серверов для размещения всех сервисов.

При решении данной задачи с использованием данного алгоритма сервисы сортируются по убыванию весов Ram, Hdd, Cpu и применяется алгоритм BF. Список серверов остается неотсортированным. В произвольном порядке размещаем сервисы по следующему правилу: первый сервис размещаем на первом сервере. На k-м шаге размещаем k-й сервис. Находим частично заполненные сервера, удовлетворяющие следующим ограничениям:

,

,

,

,

,

,

т.е. сервера, где достаточно свободного места для k-го сервиса и выбираем среди них наиболее заполненный. Если таких нет, то берем новый пустой контейнер и помещаем k-й предмет в него.

Результаты работы алгоритма представлены в таблице 2.1 и графике (рисунок 5):

Таблица 2.1. Зависимость количества свободных серверов от количества размещаемых сервисов

|  |  |
| --- | --- |
| Алгоритм BFD | |
| Кол-во сервисов | % свободных серверов |
| 0 | 100 |
| 5 | 60 |
| 10 | 40 |
| 15 | 20 |
| 20 | 0 |
| 25 | 0 |
| 30 | 0 |
| 35 | 0 |
| 40 | 0 |
| 45 | 0 |
| 50 | 0 |

Рисунок 2.7. График зависимости количества свободных серверов от количества размещаемых сервисов

## 2.5 Реализация генетического алгоритма

Решение задачи распределения сервисов на сервера с помощью генетического алгоритма представляет собой итерационную процедуру, позволяющую получать улучшенное решение на каждом шаге, до тех пор, пока не будет выполнен критерий остановки.

На каждой итерации рассматриваются сразу несколько альтернативных вариантов (особей) решения задачи. Перед запуском генетического алгоритма происходит первоначальное создание хромосом, которое заключается в случайном распределении сервисов на серверах. При распределении сервисов учитывается только ограничение на операционную систему, т.е. для каждого сервиса находится случайный элемент из множества серверов при ограничении на тип операционной системы , .

На данном этапе сервисы могут быть распределены на серверах так, что свободное место на сервере имеет отрицательное значение. Позднее, в процессе работы алгоритма это будет исправлено.

При создании исходной популяции выбирается размер популяции хромосом. Размер популяции хромосом выбран подбором. Рекомендуемый размер популяции варьируется в зависимости от задачи и в среднем составляет 30-50 хромосом [12]. Однако, для задачи, решаемой в данной работе, используется меньший размер популяции. В таблице 2.2 показано, что максимальная заполненность HDD, RAM и CPU серверов колеблется от 89 до 98, от 92 до 99 и от 92 до 99 соответственно, при увеличении числа особей в популяции с 3 до 13, однако, начиная с числа особей 9 результаты работы алгоритма практически не меняются, а лишь растет время работы алгоритма. Таким образом, подходящее число особей в популяции 9. Данные усреднены по 50 независимым запускам и представлены в таблице 2.2.

Таблица 2.2. Зависимость показателей работы генетического алгоритма от количества особей в популяции.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во особей в популяции | Время работы алгоритма | Максимальная заполненность HDD, % | Максимальная заполненность RAM, % | Максимальное использование CPU, % |
| 3 | 0,859 | 89 | 92 | 92 |
| 5 | 1,012 | 92 | 94 | 93 |
| 7 | 1,514 | 96 | 94 | 95 |
| 9 | 1,921 | 98 | 99 | 99 |
| 11 | 2,122 | 98 | 99 | 99 |
| 13 | 2.588 | 98 | 99 | 99 |

Для того, чтобы понять, двигается ли алгоритм в сторону лучшего решения, необходимо ввести критерии, по которым будет производиться сравнение решений. При решении задачи распределения сервисов на сервера необходимо занимать как можно большее свободное пространство сервера, соблюдая ограничения по свободному месту на постоянном и оперативном запоминающих устройствах. Написать что-то про экспертную оценку важности метрики? Представим данные критерии в таблице 2.3.

Таблица 2.3. Критерии сравнения решений

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Метрика | Важность | Коэффициент важности |
| Количество не занятых серверов | Больше - лучше | Высокая | 2,5 |
| количество северов с отрицательным значением свободной памяти | Меньше - лучше | Высокая | 3 |
| количество северов с положительным значением свободной памяти | Больше - лучше | Средняя | 0,2 |
| количество северов с отрицательным значением свободной оперативной памяти | Меньше - лучше | Высокая | 3 |
| количество северов с положительным значением свободной оперативной памяти | Больше - лучше | Средняя | 0,2 |

Кроме критериев, определенных в таблице 2.3, на качество решения также влияет показатель разброса количества наиболее занятых и наименее занятых серверов. Чем выше дисперсия, тем ближе решение к оптимальному. Данный показатель можно получить, рассчитав по каждому серверу отношения полезного объема, занимаемого сервисами, к общему объему сервера, а далее, получить дисперсию данных показаний.

Расчет отношения занятого пространства к общему на примере трехмерного контейнера с занятым объемом и общим объемом рассчитывается по формуле

В случае с сервером, отношение занятого пространства сервера к общему объему может быть рассчитано так:

где - количество свободной оперативной памяти, - общий объем оперативной памяти, - количество свободной памяти на постоянном запоминающем устройстве, – общий объем памяти на постоянном запоминающем устройстве, – количество неиспользованных процессорных ресурсов, – общее количество возможного использования процессора, – коэффициент важности индекса заполненности постоянной памяти сервера, – коэффициент важности индекса заполненности оперативной памяти сервера, – коэффициент важности индекса использования процессорного времени сервера.

В таблице 2.4 представлены коэффициенты заполненности сервера и их экспертные оценки.

Таблица 2.4. Коэффициенты заполненности сервера.

|  |  |  |  |
| --- | --- | --- | --- |
| Коэффициент | Метрика | Важность | Коэффициент важности |
| Коэффициент занятости HDD | Больше - лучше | Средняя | 0,2 |
| Коэффициент занятости RAM | Больше - лучше | Высокая | 0,4 |
| Коэффициент занятости CPU | Больше - лучше | Высокая | 0,4 |

Формула расчета дисперсии в теории вероятностей имеет вид:

*,*

то есть дисперсия — это математическое ожидание отклонений от математического ожидания. На практике при анализе выборок математическое ожидание, как правило, не известно. Поэтому вместо него используют оценку – среднее арифметическое. Расчет дисперсии производят по формуле:

Где  –дисперсия,  – отдельные значения, – среднее арифметическое по выборке.

В конечном итоге, формула расчета значения функции приспособленности для каждой хромосомы из популяции задана таким образом, чтобы характеризовать параметр, по которому проводится поиск оптимального решения.

, где

- количество не занятых серверов,

– количество северов с отрицательным значением свободной памяти,

- количество северов с положительным значением свободной памяти,

- количество северов с отрицательным значением свободной оперативной памяти,

­- количество северов с положительным значением свободной оперативной памяти,

– дисперсия,

= 2.5, = 3, = 0.2, = 3, = 0.2 - коэффициенты поощрения или штрафа.

В качестве поощрения за повышение количества не занятых серверов используется коэффициент , за превышение свободного места на диске и оперативной памяти назначается штраф и , а за освобождение пространства на диске и оперативной памяти назначается поощрение и .

Функция приспособленности отвечает основным требованиям, а именно:

* распределение значений функции совпадает с распределением реального качества решений. Чем лучше решение, тем значение функции приспособленности выше;
* не имеет плоских участков;
* не требует больших вычислительных ресурсов.

Для создания новой популяции используется селекция усечением, т.е. отбираются особи с наивысшим значением функции приспособленности.

Метод скрещивания – многоточечное скрещивание. На этом этапе извлекается список сервисов, расположенных на первой родительской хромосоме . Элементы этого списка совпадают с элементами списка на второй родительской хромосоме .

Далее, итеративно выполняется перебор элементов списка сервисов, сервис с четным индексом располагается в хромосоме-потомке на том же сервере, на котором он расположен на первой родительской хромосоме , где и тип ОС . Сервис с нечетным индексом располагается на хромосоме-потомке на том же сервере, на котором он расположен на второй родительской хромосоме где и тип ОС . (рис.2.8).



Рисунок 2.8 – схема скрещивания хромосом

Во второй хромосоме-потомке сервис с четным индексом располагается на том сервере, на котором он расположен на второй родительской хромосоме , и тип ОС ., а сервис с нечетным индексом располагается на том сервере, на котором он расположен на первой родительской хромосоме , где и тип ОС . (рис. 2.9).



Рисунок 2.9 – схема скрещивания хромосом

При таком скрещивании возможен выход за пределы памяти и др. метрик, но этот недостаток компенсируется при приближении решения к оптимальному.

После выполнения операции скрещивания выполняется операция мутации, т.е. случайного изменения полученных в результате скрещивания хромосом. Операция мутации применяется с некоторой вероятностью, здесь используется аналогия с природой, т.к. в природе мутируют не все живые организмы, являющиеся потомками. Применительно к задаче размещения сервисов на серверах выбирается хромосома, в которой случайный сервис из случайного сервера и перемещается на другой случайный сервер. Ограничения по ОС соблюдаются.

Вероятность применения оператора мутации влияет на вырождаемость популяции и нахождение алгоритмом решения с наименьшей погрешностью. С целью определить погрешность работы алгоритма, произведено сравнение значения функции приспособленности решений, полученных в результате работы алгоритма с различными значениями вероятности мутации со значением функции приспособленности оптимального решения с использованием данных из тестовой выборки (приложение А). Сравнительные данные приведены в таблице 2.5. Также, на рисунке 2.10 приведен график зависимости точности результата работы алгоритма от вероятности мутации.

Таблица 2.5. Результаты работы генетического алгоритма с различной вероятностью применения оператора мутации.

|  |  |  |  |
| --- | --- | --- | --- |
| Вероятность мутации | Реальное значение функции приспособленности | Функция приспособленности оптимального решения | Погрешность, % |
| 0,05 | -12,58 | 83,58 | 115,05 |
| 0,1 | 59,14 | 83,58 | 29,24 |
| 0,2 | 68,88 | 83,58 | 17,59 |
| 0,3 | 79,47 | 83,58 | 4,92 |
| 0,4 | 79,47 | 83,58 | 4,92 |
| 0,5 | 79,47 | 83,58 | 4,92 |
| 0,6 | 79,47 | 83,58 | 4,92 |
| 0,7 | 79,47 | 83,58 | 4,92 |
| 0,8 | 79,47 | 83,58 | 4,92 |
| 0,9 | 79,47 | 83,58 | 4,92 |
| 1 | 79,47 | 83,58 | 4,92 |

Рисунок 2.10. Зависимость точности результата работы алгоритма от вероятности мутации

Согласно таблице 2.5 погрешность работы алгоритма не меняется при значении вероятности мутации в диапазоне от 0,3 до 1. Примем значение вероятности мутации равным 0,3.

Алгоритм останавливается, если функция приспособленности не изменяется на протяжении 100 итераций.

В процессе работы алгоритма не выявлено признаков преждевременной сходимости и вырождения популяции, все особи популяции различаются между собой.

В таблице 2.6 представлены результаты работы алгоритма. На рисунке 2.11 представлен график зависимости количества свободных серверов от количества размещаемых сервисов.

Таблица 2.6. Зависимость количества свободных серверов от количества размещаемых сервисов

|  |  |
| --- | --- |
| Генетический алгоритм | |
| Кол-во сервисов | % свободных серверов |
| 0 | 100 |
| 5 | 60 |
| 10 | 60 |
| 15 | 60 |
| 20 | 60 |
| 25 | 60 |
| 30 | 40 |
| 35 | 40 |
| 40 | 0 |
| 45 | 0 |
| 50 | 0 |

Рисунок 2.11. График зависимости количества свободных серверов от количества размещаемых сервисов

## 2.6 Реализация алгоритма имитации отжига

Как сказано в пункте 2.2, метод отжига служит для поиска глобального минимума некоторой функции , заданной для x из некоторого пространства , дискретного или непрерывного. Элементы множества представляют собой состояния воображаемой физической системы (энергетические уровни) – в нашем случае состоянием будет вариант распределения сервисов на сервера. В данном случае энергия системы это значение функции

, где

- количество не занятых серверов,

– количество северов с отрицательным значением свободной памяти,

- количество северов с положительным значением свободной памяти,

- количество северов с отрицательным значением свободной оперативной памяти,

­- количество северов с положительным значением свободной оперативной памяти,

- показатель разброса количества наиболее занятых и наименее занятых серверов, а коэффициенты

= 2.5, = 3, = 0.2, = 3, = 0.2 – коэффициенты штрафа и поощрения.

В данном случае рассчитывается так же, 2.5 при расчете фитнесс функции генетического алгоритма, за тем исключением, что значение берется с отрицательным знаком.

В каждый момент предполагается заданной температура системы , уменьшающаяся с течением времени. При уменьшении температуры , следующее состояние системы выбирается в соответствии с порождающим семейством вероятностных распределений , которое при фиксированных и задает новое местоположение случайного элемента из множества сервисов на случайном сервере в пространстве . После генерации нового состояния система с вероятностью переходит к следующему шагу в состояние , в противном случае, процесс генерации повторяется. Здесь означает приращение функции энергии, т.е. разность значений функции энергии текущего и предыдущего состояний . Если , то вероятность перехода в новое состояние считается равной 1. Таким образом, если новое состояние дает лучшее значение оптимизируемой функции, то переход в это состояние произойдет в любом случае. Если же , то переход осуществляется с вероятностью . Понижение температуры осуществляется по закону Больцмановского отжига .

В таблице 2.7 представлены результаты работы алгоритма имитации отжига. На рисунке 2.12 представлен график зависимости количества свободных серверов от количества размещаемых сервисов.

Таблица 2.7. Зависимость количества свободных серверов от количества размещаемых сервисов

|  |  |
| --- | --- |
| Алгоритм имитации отжига | |
| Кол-во сервисов | % свободных серверов |
| 0 | 100 |
| 5 | 60 |
| 10 | 60 |
| 15 | 60 |
| 20 | 60 |
| 25 | 60 |
| 30 | 40 |
| 35 | 40 |
| 40 | 0 |
| 45 | 0 |
| 50 | 0 |

Рисунок 2.12. Зависимость количества свободных серверов от количества размещаемых сервисов

## 2.7 Анализ работы алгоритмов и выбор наиболее подходящего алгоритма

С целью сравнения результатов алгоритмов и выбора наиболее подходящего из них, данные алгоритмы были проверены на тестовом наборе данных (приложение А).

Нужна какая-то экспертная группа и методика

Для того, чтобы сравнить вышеописанные алгоритмы, необходимо определить критерии их сравнения. Первый критерий – время работы алгоритма. Данный критерий имеет среднюю важность и ограничение работы до 10 минут. Следующий критерий – заполненность серверов. Заполненность серверов может быть рассчитана как убывающий ряд, каждое значение которого является процентным соотношением кол-ва заполненных серверов к общему кол-ву серверов при увеличении кол-ва сервисов. Критерий имеет высокую важность, чем выше значение заполненности сервера, тем более полно используется его свободное место. Если сервер заполнен полностью, значение заполненности равно единице. Вероятность ошибочного решения рассчитывается по 50 независимым запускам. Решение считается ошибочным, если в результате работы алгоритма полученное решение не отвечает ограничениям, установленным в пункте 1 главы 2.

Критерии выбора и их экспертная оценка представлены в таблице 2.8.

Таблица 2.8. Критерии сравнения алгоритмов комбинаторной оптимизации.

|  |  |  |  |
| --- | --- | --- | --- |
| Название критерия | Важность | Единицы измерения | Метод расчета |
| Время работы алгоритма | Средняя, время работы до 10 минут | Секунды | Измерения времени работы с момента запуска до получения решения |
| Заполняемость | Высокая | - | процентным соотношением кол-ва заполненных серверов к общему кол-ву серверов при увеличении кол-ва сервисов |
| Вероятность ошибочного решения | Высокая | - | Отношение кол-ва ошибочных решений к общему числу запусков |

В качестве исходных данных было использовано 5 виртуальных машин (серверов) и 50 сервисов, которые необходимо распределить по данным серверам.

В таблице 2.9 представлены результаты сравнения времени работы алгоритмов и рассчитана вероятность ошибочного решения.

Таблица 2.9. Время работы алгоритма и вероятность ошибочного решения для различных алгоритмов.

|  |  |  |
| --- | --- | --- |
| Название алгоритма | Время работы алгоритма, с | Вероятность ошибочного решения |
| Эвристический алгоритм | 0,0022 | 0 |
| Генетический алгоритм | 1,2631 | 0,01 |
| Алгоритм имитации отжига | 0,2365 | 0,07 |

Зависимость количества свободных серверов от количества размещаемых сервисов (таблица 2.10, рисунок 2.13) показывает, что генетический алгоритм и алгоритм имитации отжига одинаково заполняют сервера при последовательном увеличении количества сервисов от 0 до 50.

Таблица 2.10. Зависимость количества свободных серверов от количества размещаемых сервисов

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | % свободных серверов | | | | | | | | | | |
| Кол-во сервисов | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| Эвристический алгоритм | 100 | 60 | 40 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Генетический алгоритм | 100 | 60 | 60 | 60 | 60 | 60 | 40 | 40 | 0 | 0 | 0 |
| Алгоритм имитации отжига | 100 | 60 | 60 | 60 | 60 | 40 | 40 | 40 | 0 | 0 | 0 |

Рисунок 2.13. Зависимость количества свободных серверов от количества размещаемых сервисов

Ниже в таблицах 2.11-2.13 представлены данные отдельно по заполненности Hdd, Ram, Cpu. На графиках (рисунки 2.14 - 2.19) показано, что использование свободного пространства серверов тем выше, чем более круто падает кривая.

Таблица 2.11. Заполненность Hdd для серверов.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя сервера | Заполненность Hdd, % | | |
| Алгоритм BFD | Генетический алгоритм | Алгоритм имитации отжига |
| WIN\_1 | 94 | 99 | 100 |
| WIN\_2 | 87 | 95 | 100 |
| WIN\_3 | 84 | 66 | 60 |
| LIN\_4 | 61 | 78 | 95 |
| LIN\_5 | 61 | 45 | 28 |

Рисунок 2.14. Заполненность HDD Windows серверов

Рисунок 2.15. Заполненность HDD Linux серверов

Таблица 2.12. Заполненность Ram для серверов.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя сервера | Заполненность Ram, % | | |
| Алгоритм BFD | Генетический алгоритм | Алгоритм имитации отжига |
| WIN\_1 | 85 | 93 | 99 |
| WIN\_2 | 51 | 58 | 58 |
| WIN\_3 | 79 | 66 | 48 |
| LIN\_4 | 46 | 35 | 81 |
| LIN\_5 | 46 | 56 | 10 |

Рисунок 2.16. Заполненность Ram Windows серверов

Рисунок 2.17. Заполненность Ram Linux серверов

Таблица 2.13. Заполненность Cpu для серверов.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя сервера | Использование Cpu, % | | |
| Алгоритм BFD | Генетический алгоритм | Алгоритм имитации отжига |
| WIN\_1 | 100 | 98 | 100 |
| WIN\_2 | 50 | 49 | 38 |
| WIN\_3 | 22 | 24 | 33 |
| LIN\_4 | 14 | 21 | 18 |
| LIN\_5 | 15 | 13 | 11 |

Рисунок 2.18. Использование Cpu Windows серверов

Рисунок 2.19. Использование Cpu Linux серверов

Принимая во внимание показатели работы вышеописанных алгоритмов, представляется возможным сделать вывод, что наиболее подходящим для решения задачи распределения сервисов по виртуальным машинам (серверам) является…

# ПРИЛОЖЕНИЕ А.

Таблица 1. Характеристики серверов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Характеристики | | | |
|  | Имя сервера | Тип операционной системы | Свободное место на жестком диске, Гб | Количество оперативной памяти, Гб |
| 1 | Server\_1 | Windows | 142 | 34 |
| 2 | Server \_2 | Windows | 54 | 22 |
| 3 | Server \_3 | Windows | 66 | 20 |
| 4 | Server \_4 | Windows | 36 | 20 |
| 5 | Server \_Linux | Linux | 36 | 20 |

Таблица 1. Характеристики сервисов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Характеристики | | | |
|  | Имя сервиса | Тип операционной системы | Количество занимаемого места на жестком диске, Гб | Количество необходимой оперативной памяти, Гб |
| 1 | Сервис\_1 | Windows | 5 | 1 |
| 2 | Сервис\_2 | Windows | 12 | 2 |
| 3 | Сервис\_3 | Windows | 3 | 1 |
| 4 | Сервис\_4 | Windows | 7 | 2 |
| 5 | Сервис\_Linux\_1 | Linux | 8 | 2 |
| 6 | Сервис\_5 | Windows | 2 | 1 |
| 7 | Сервис\_6 | Windows | 10 | 2 |
| 8 | Сервис\_7 | Windows | 13 | 3 |
| 9 | Сервис\_8 | Windows | 4 | 2 |
| 10 | Сервис\_Linux\_2 | Linux | 2 | 3 |
| 11 | Сервис\_9 | Windows | 5 | 1 |
| 12 | Сервис\_10 | Windows | 10 | 2 |
| 13 | Сервис\_11 | Windows | 3 | 1,5 |
| 14 | Сервис\_12 | Windows | 8,5 | 2 |
| 15 | Сервис\_Linux\_3 | Linux | 11 | 2 |
| 16 | Сервис\_13 | Windows | 4 | 1 |
| 17 | Сервис\_14 | Windows | 12 | 2 |
| 18 | Сервис\_15 | Windows | 8 | 1,5 |
| 19 | Сервис\_16 | Windows | 6,3 | 2 |
| 20 | Сервис\_Linux\_4 | Linux | 1 | 2 |
| 21 | Сервис\_17 | Windows | 1 | 0,5 |
| 22 | Сервис\_18 | Windows | 0,5 | 0,2 |
| 23 | Сервис\_19 | Windows | 1 | 0,2 |
| 24 | Сервис\_20 | Windows | 0,3 | 0,12 |
| 25 | Сервис\_Linux\_5 | Linux | 0,1 | 0,1 |
| 26 | Сервис\_21 | Windows | 5 | 1 |
| 27 | Сервис\_22 | Windows | 12 | 2 |
| 28 | Сервис\_23 | Windows | 3 | 1 |
| 29 | Сервис\_24 | Windows | 7 | 2 |
| 30 | Сервис\_Linux\_6 | Linux | 8 | 2 |
| 31 | Сервис\_25 | Windows | 2 | 1 |
| 32 | Сервис\_26 | Windows | 10 | 2 |
| 33 | Сервис\_27 | Windows | 13 | 3 |
| 34 | Сервис\_28 | Windows | 4 | 2 |
| 35 | Сервис\_Linux\_7 | Linux | 2 | 3 |
| 36 | Сервис\_29 | Windows | 5 | 1 |
| 37 | Сервис\_30 | Windows | 12 | 2 |
| 38 | Сервис\_31 | Windows | 3 | 1,5 |
| 39 | Сервис\_32 | Windows | 8,5 | 2 |
| 40 | Сервис\_Linux\_8 | Linux | 11 | 2 |
| 41 | Сервис\_33 | Windows | 4 | 1 |
| 42 | Сервис\_34 | Windows | 12 | 2 |
| 43 | Сервис\_35 | Windows | 8 | 1,5 |
| 44 | Сервис\_36 | Windows | 6,3 | 2 |
| 45 | Сервис\_Linux\_9 | Linux | 1 | 2 |
| 46 | Сервис\_37 | Windows | 1 | 0,5 |
| 47 | Сервис\_38 | Windows | 0,5 | 0,2 |
| 48 | Сервис\_39 | Windows | 1 | 0,2 |
| 49 | Сервис\_40 | Windows | 0,3 | 0,12 |
| 50 | Сервис\_Linux\_10 | Linux | 0,1 | 0,1 |

Данные в формате json

[

{

"Name" : "Server\_1",

"Os" : "Windows",

"HddFull" : 142,

"HddFree" : 142,

"RamFull" : 34,

"RamFree" : 34,

"CpuFull" : 100,

"CpuFree" : 100,

"Services" : []

},

{

"Name" : "Server\_2",

"Os" : "Windows",

"HddFull" : 54,

"HddFree" : 54,

"RamFull" : 22,

"RamFree" : 22,

"CpuFull" : 100,

"CpuFree" : 100,

"Services" : []

},

{

"Name" : "Server\_3",

"Os" : "Windows",

"HddFull" : 66,

"HddFree" : 66,

"RamFull" : 20,

"RamFree" : 20,

"CpuFull" : 100,

"CpuFree" : 100,

"Services" : []

},

{

"Name" : "Server\_4",

"Os" : "Windows",

"HddFull" : 36,

"HddFree" : 36,

"RamFull" : 20,

"RamFree" : 20,

"CpuFull" : 100,

"CpuFree" : 100,

"Services" : []

},

{

"Name" : "Server\_Linux",

"Os" : "Windows",

"HddFull" : 36,

"HddFree" : 36,

"RamFull" : 20,

"RamFree" : 20,

"CpuFull" : 100,

"CpuFree" : 100,

"Services" : []

}

]

# СПИСОК ЛИТЕРАТУРЫ

1. Лекции по дискретной математике Федеральное государственное бюджетное учреждение науки Институт математики им. С. Л. Соболева Сибирского <http://www.math.nsc.ru/LBRT/k5/TPR/lec6.pdf>
2. Стр 221-224 Library of Congress Cataloging-in-Publication Data: Martello, Silvano. Knapsack problems : algorithms and computer implementations 296 стр 1990 год <http://www.or.deis.unibo.it/kp/KnapsackProblems.pdf>
3. Батищев Д.И., Неймарк Е.А., Старостин Н.В. Применение генетических алгоритмов к решению задач дискретной оптимизации. Учебно-методический материал по программе повышения квалификации «Информационные технологии и компьютерное моделирование в прикладной математике». Нижний Новгород, 2007, 85 с.
4. Растригин Л. А. Случайный поиск — специфика, этапы истории и предрассудки. Вопросы кибернетики. Вып. 33 (1978), с. 3–16.
5. Holland J. H. Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press. 1975.
6. Rechenberg I. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der Biologischen Information, Freiburg: Fromman, 1973.
7. Еремеев А.В. Разработка и анализ генетических и гибридных алгоритмов для решения задач дискретной оптимизации. Дисс. канд.физ.-мат.наук. Омск, 2000.
8. Goldberg D. E. Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison-Wesley. 1989.
9. ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ КОНКУРЕНТНОЙ ЗАДАЧИ О P-МЕДИАНЕ. Е.В. Алексеева, А.В. Орлов. Институт математики им. С.Л. Соболева СО РАН, Новосибирский Государственный Университет, Новосибирск
10. Батищев Д.И., Неймарк Е.А., Старостин Н.В. Применение генетических алгоритмов к решению задач дискретной оптимизации. Учебно-методический материал по программе повышения квалификации «Информационные технологии и компьютерное моделирование в прикладной математике». Нижний Новгород, 2007, 85 с.
11. Генетический алгоритм: теория и практика: учеб. пособие / М. В. Бураков. – СПб.: ГУАП, 2008. – 164 с.: ил. <http://window.edu.ru/resource/205/80205/files/Burakov_Gen_algoritm_teor_i_prakt.pdf>
12. Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 [3] с
13. Jeong C., Kim M. Fast Parallel Simulated Annealing for Traveling Salesman Problem on SIMD Machines with Linear Interconnections // Parallel Computing. 17. 1991. P. 221228.
14. Yao X. Call Routing by Simulated Annealing // International Journal of Electronics. Oct. 1995.
15. Нужнов Е.В., Барлит А.В. Трехмерная упаковка несвязных элементов на основе эвристических процедур. – Таганрог: Изд-во ТРТУ, 2002. – 23 с.
16. Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H., and Teller E. Equation of State Calculations by Fast Computer Machines // J. Chemical Physics. 21. 6. June. 1953. P. 10871092.
17. *Базаров И. П., Геворкян Э. В., Николаев П. Н.* Термодинамика и статистическая физика. Теория равновесных систем. — М.: МГУ, 1986. — 312 с.
18. *УРАЛЬСКИЙ НИКОЛАЙ БОРИСОВИЧ, СИЗОВ ВАЛЕРИЙ АЛЕКСАНДРОВИЧ, КАПУСТИН НИКОЛАЙ КЛЕМЕНТЬЕВИЧ.* [Оптимизация вычислительного процесса фитнесс-функции генетического алгоритма в распределённых системах обработки данных](http://cyberleninka.ru/article/n/optimizatsiya-vychislitelnogo-protsessa-fitness-funktsii-geneticheskogo-algoritma-v-raspredelyonnyh-sistemah-obrabotki-dannyh) // Интернет-журнал Науковедение. — 2015-01-01. — Т. 7, вып. 6 (31).
19. [А. В. Смирнов. О задаче упаковки в контейнеры. УМН, 1991, том 46, выпуск 4(280), страницы 173–174.](http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=rm&paperid=4649&option_lang=rus)
20. *Томас Х. Кормен и др.* **Глава 34. NP-полнота** // Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2006. — 1296 с. — [ISBN 0-07-013151-1](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/0070131511).