**Содержание**

[Введение 4](#_Toc358277578)

[1 Описание предметной области 6](#_Toc358277579)

[1.1 Многоообразие задач раскроя- упаковки и поиска маршрута 7](#_Toc358277580)

[1.2 Классификация задач раскроя- упаковки 7](#_Toc358277581)

[2 Методы решения задач раскроя и упаковки 11](#_Toc358277582)

[2.1 Подход линейного целочисленного программирования. 11](#_Toc358277583)

[2.2 Двухэтапны подходы на базе линейного и линейного целочисленного программирования. 12](#_Toc358277584)

[2.3 Метод последовательного уточнения оценок (SVC) 13](#_Toc358277585)

[2.4 Алгоритмы типа On-line 14](#_Toc358277586)

[2.4.1 Алгоритм «Следующий подходящий» (NF) 15](#_Toc358277587)

[2.4.2 Алгоритм «Первый подходящий» (FF) 16](#_Toc358277588)

[2.4.3 Алгоритм «Наилучший подходящий» (BF) 17](#_Toc358277589)

[2.4.4 Алгоритм «Первый подходящий с упорядочиванием» (FFD) 18](#_Toc358277590)

[2.5 Метод генерации столбцов 18](#_Toc358277591)

[2.6 Генетический алгоритм 20](#_Toc358277592)

[2.6.1 Генетический алгоритм, применительно к задаче раскроя 24](#_Toc358277593)

[2.6.2 Генетический алгоритм с алгоритмами «нижний левый» и SVC 25](#_Toc358277594)

[2.7 Муравьиный алгоритм 27](#_Toc358277595)

[3 Обзор аналогичных программных систем и выбор инструментального средства разработки 31](#_Toc358277596)

[3.1 Описание аналогичных программных систем 31](#_Toc358277597)

[3.1.1 Система «Техтран-Раскрой» 31](#_Toc358277598)

[3.1.2 Программа «Астра Раскрой» 32](#_Toc358277599)

[3.1.3 Сапр «Интех-Раскрой» 33](#_Toc358277600)

[3.1.4 Программа «2D-Place» 34](#_Toc358277601)

[3.1.5 Сапр «Сириус» 35](#_Toc358277602)

[3.1.6 Программа «T-Flex/Раскрой» 37](#_Toc358277603)

[3.1.7 САПР «AccuMark» 38](#_Toc358277604)

[3.1.8 Программа «OptiTex» 39](#_Toc358277605)

[4 Разрабонтанное приложение 41](#_Toc358277606)

[Список использованных источников 48](#_Toc358277607)

[Приложение А. Текст программы 52](#_Toc358277608)

**Введение**

В условиях современного развития производства и технологий, возникает потребность в использовании технологических процессов и методов, по средством которых минимизируются затраты. В качестве затрат выступают не только денежные средства, но и время выполнения, материалы, пройденное расстояние. Минимизации подобных параметров можно добиться внедрением различных методов матеметического программирования.

Задачи снижения уровня затрат материалов и времени работы оборудования играют важную роль в технологических процессах, и отражаются на эфеективности производства в целом.

Задача снижения уровня затрат материалов рассматривается для процесса раскроя материала, а следовательно, попадает в класс задач раскроя-упаковки, которые также решают проблему комплектования грузов в контейнеры.

Вторая задача здесь рассматривается для обхода режущего оборудования по линиям раскроя материала, и она попадает под класс транспортных задач, так как от длины пути напрямую и зависит время и эффективность его работы.

По сложности данные задачи относятся к классу NP-трудных задач оптимизации, т.е. к таким задачам, для которых пока не существует методов и алгоритмов, находящих точное решение за поли­номиальное время.

Транспортные задачи — специальный класс задач линейного программирования. Эти модели часто описывают перемещение (перевозку) какого- либо товара из пункта отправления (исходный пункт, например место производства) в пункт назначения (склад, магазин, грузохранилище). Назначение транспортной задачи — определить объем перевозок из пунктов отправления в пункты назначения с минимальной суммарной стоимостью перевозок. При этом должны учитываться ограничения, налагаемые на объемы грузов, имеющихся в пунктах отправления (предложения), и ограничения, учитывающие потребность грузов в пунктах назначения (спрос). В транспортной модели предполагается, что стоимость перевозки по какому-либо маршруту прямо пропорциональна объему груза, перевозимого по этому маршруту. В общем случае транспортную модель можно применять для описания ситуаций, связанных с управлением запасами, управлением движением капиталов, составлением расписаний, назначением персонала и др.

В нашем случае искомый маршрут – маршрут движения режущего оборудованием, осуществляющего раскрой материала по составленной карте раскроя.

Целью данной работы является реализация автоматизации процессов рационального раскроя материала и поиска рационального маршрута движения режущего инструмента.

В соответствии с поставленной целью можно выделить следующие этапы разработки:

1. анализ функций конечного пользователя;
2. анализ задач раскроя-упаковки и поиска маршрута;
3. анализ методов решения;
4. разработка сценариев диалога пользователя с системой;
5. опеределение входной и выходной информации;
6. разработка интерфейса системы;
7. программная реализация алгоритмов.

1 Описание предметной области

Под задачами раскроя-упаковки понимается широкий класс проблем, допускающих различное толкование [6]. Общим является наличие двух групп объектов. Между элементами этих групп устанавливается и оценивается соответствие. Цель решения данной задачи заключается в минимизации расхода материала.

В первые задачу раскроя-упаковки была рассмотрена русским сатематиком Чебышевым П.Л. в его докладе 1878 г. «О кройке одежды»[8]. В этой работе Чебышев рассмотрел вопрос об оптимальном покрытии кривых поверхностей плоскими выкройками из ткани. Начиная с 1933 г. Регулярно стали появляться в печати работы, посвященные раскрою материалов. Так в конце 30-х – начале 40-х годов фундаментальные исследования в области рационального раскроя были выполнены Л. В. Канторовичем и В. А. Залгаллером. Они показали, что для расчета оптимального раскроя могут быть использованы методы линейного программриования, а в 1951 г. Данциг, также показал связь между задачами раскроя и теорией линейного программирования. В 1961 г. Гилмори и Гомори решили с помощью метода линейного программирования проблему одномерного раскроя, а в 1965 г. они предложили метод решения двумерного гильотинного раскроя. В 1966 г. для решения задач одномерного и двумерного раскроев они использовали динамическое программирование. В 60-х годах методам решения задач фигурного раскроя были посвящены работы Л. Б. Беляковой. В 1977 г. Кристофидес использовал дерево решений для нахождения оптимального решения для двумерного раскроя. Это было первое применение эвристики для решения двумерного раскроя. Качественную типологию в области раскроя-упаковки провел в 1991 Дикхова.

Под рамки задач поиска оптимального маршрута также подходит множество проблем. Ключивым в данных задачах является наличие объектов, которые нужно обойти и субъекта, который будет совершать данный обход, и требуется составить порядок обхода, минимизирую его вес.

В данной работе рассматривается минимизация раектории движения режущего инструмента, которая состоит из следующих элементов[1]:

1. внешних контуров вырезаемых деталей;
2. внутренних контуров;
3. траекторий, связывающих смежные контуры;
4. траекторий переходов инструмента в выключенном состоянии от одной точки врезки к другой.

1.1 Многоообразие задач раскроя- упаковки и поиска маршрута

В литературе задачи раскроя-упаковки встречаются с различными названиями:

1. задача раскроя-упаковки;
2. задача раскроя запаса;
3. задача плотного размещения геометрических объектов в заданной области;
4. задача упаковки в контейнеры;
5. задача загрузки транспорта;
6. задача загрузки рюкзака;
7. задача составления расписания;
8. и т.д..

А задачи поиска рационального маршрута встречаются в следующих формулировках:

1. задача коммивояжера;
2. задача нахождения кратчайшего пути в графе;
3. задача поиска самого выгодного маршрута;
4. задача поиска самого дешевого маршрута;
5. и т.д..

1.2 Классификация задач раскроя- упаковки

Приведем классификацию задач раскроя-упаковки[15]:

Одномерные задачи раскроя и упаковки:

1. загрузка рюкзака (Knapsack Packing, KP);
2. одномерный раскрой (1 Dimensional Cutting Stock, 1DCS). Материал для раскроя на заготовки заданных размеров поступает в виде одинаковых стержней (форм). Требуется раскроить стержни на заготовки с наименьшими затратами материала;
3. одномерная упаковка (1 Dimensional Bin Packing, 1DBP). частный случай задачи раскроя, каждая заготовка требуется в единственном экземпляре.

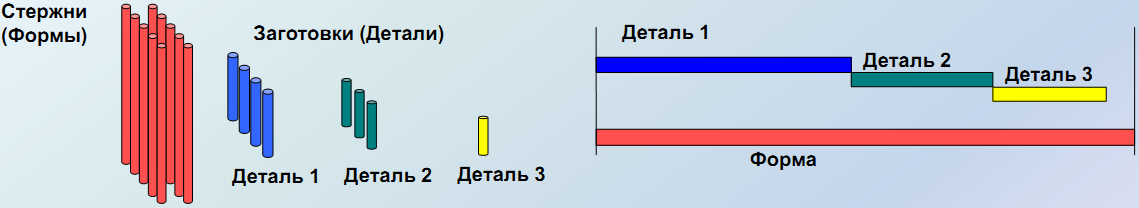


Рис. 1.1 – Одномерная упаковка

Двумерные задачи раскроя-упаковки:

1. Задачи прямоугольной упаковки:
2. упаковка прямоугольников в полосу (2D Strip Packing Problem, 2DSP);

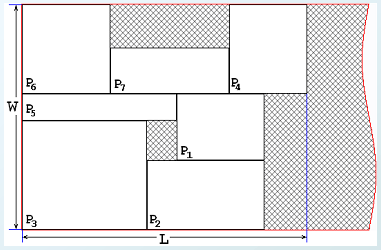


Рис. 1.3 – Упаковка прямоугольников в полосу

1. упаковка прямоугольников в контейнеры (2D Bin Packing Problem, 2DBP);

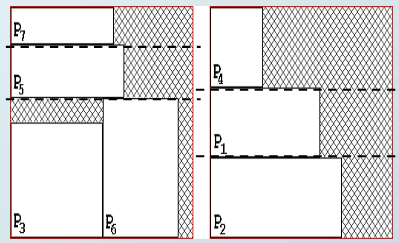


Рис. 1.4 – Упаковка прямоугольников в контейнеры

1. упаковка прямоугольников в открытую область(квадрант) (2D Area Packing Problem, 2DAP).

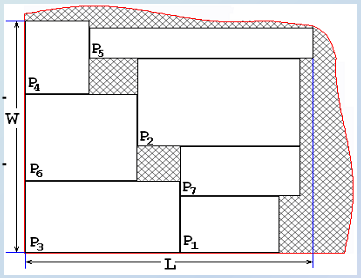


Рис. 1.5 – Упаковка прямоугольников в открытую область

1. Задачи гильотинного раскроя:
2. гильотинный раскрой полубесконечной полосы (2 Dimensional Guillotine Strip Cutting, 2DGSC);
3. гильотинный раскрой листов (2 Dimensional Guillotine Bin Cutting, 2DGBC).

Задачи трехмерной упаковки:

1. размещение ящиков в полубесконечный контейнер (с одной незафиксированной стороной) (3 Dimensional Strip Packing Problem, 3DSP);
2. размещение ящиков в ограниченные контейнеры (3 Dimensional Bin Packing Problem, 3DBP).

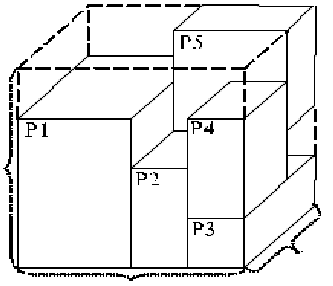


Рис. 1.6 – Размещение ящиков в ограниченные контейнеры

Задачи фигурного раскроя-упаковки (нестинга):

1. Плоские фигуры.

Задачи максимального покрытия:

1. покрытие стержней (1 Dimensional Skiving Stock Problem, 1DSSP). требуется найти план полного покрытия максимального количества форм исходными деталями;



Рис. 1.7 – Покрытие стержней

1. покрытие листов (2 Dimensional Skiving Stock Problem, 2DSSP).

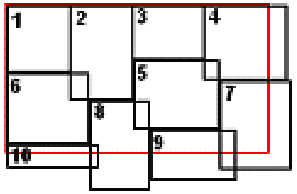


Рис. 1.8 – Покрытие листов

N-мерная упаковка:

1. упаковки N-мерных параллелепипедов в N-мерную полубесконечную полосу.

2 Методы решения задач раскроя и упаковки

2.1 Подход линейного целочисленного программирования.

Материал для раскроя на заготовки заданных размеров поступает в виде одинаковых стержней (полос)[9]. Требуется раскроить стержни на заготовки с наименьшими затратами материала. Задача линейного раскроя (1DCSP) определяется следующим информационным вектором:

, (1)

где L – длина раскраиваемого материала; m – количество различных заготовок; , i= , - длины (веса) заготовок; , - требуемое количество заготовок. Вектор описывает шаблон раскроя j с компонентами , указывающим в нем количество заготовок типа . Матрица , где n – количество шаблонов, называется раскройной матрицей;  означает допустимый раскрой, если

 (2)

Обозначим через  - интенсивность применения раскроя j. Проблема планирования оптимального раскроя материала сводится к минимизации функции

 (3)

Если считать, что все векторы  заданы, то 1DCSP является задачей линейного целочисленного программирования. Однако количество n различных шаблонов раскроя очень велико, поэтому столбцы матрицы А задаются неявно через условия (2). Они генерируются на каждой итерации линейного программирования путем решения решения вспомогательной задачи динамического программирования. Это позволило решать проблемы раскроя с использованием линейного программирования и находить непрерывные значения компонент вектора x и функции . С помощью этого подхода Л. В. Канторович и В. А. Залгаллер и предложили решать задачи линейного и гильотинного раскроя в условиях массового или крупносерийного производства.

В 1985 г. Маркоте сформулировал важные свойства целочисленного решения для задачи линейного раскроя:

1. wелочисленные свойства (Integer Property, IP), если целочисленный оптимум  совпадает с оптимальным решением  задачи непрерывной релаксации;
2. свойство округления до целого вверх (Integer Round-up Property, IRUP), если=[ ];
3. модифицированное свойство округления до целого вверх (MIRUP), если ≤[ ]+1;

Относительно этих свойств высказано предположение, что стандартная одномерная задача раскроя обладает свойством MIRUP. Однако это предположение до сих пор не опровергнуто, но и не доказано.

2.2 Двухэтапны подходы на базе линейного и линейного целочисленного программирования.

На базе подхода линейного целочисленного программирования был предложен двухэтапный процесс расчета целочисленного раскроя для случаев единичного или мелкосерийного производства[9]. На первом этапе с помощью линейного программирования решается задача непрерывной релаксации. Найденное оптимальное значение  округляется сверху до ближайшего целого , а полученные значения  округляются до целого снизу. При этом  служит нижней границей оптимума для исходной задачи и формулируется остаточная задача (Surplus Problem, SP) из недополученных заготовок. Для решения SP на втором этапе применяется эвристика.

2.3 Метод последовательного уточнения оценок (SVC)

Развитие двухэтапного подхода привело к использованию отсекающих плоскостей Гомори[9]. При добавлении сечений непрерывное решение становится все ближе к целочисленному оптимуму. Для генерации новых сечений в случае, когда имеется только базисная матрица, используется процедура округления Хватала-Гомори (Chvatal-Gomory Rounding Procedure, C-GRP). На втором этапе используется эвристика – метод последовательного уточнения оценок (SVC).

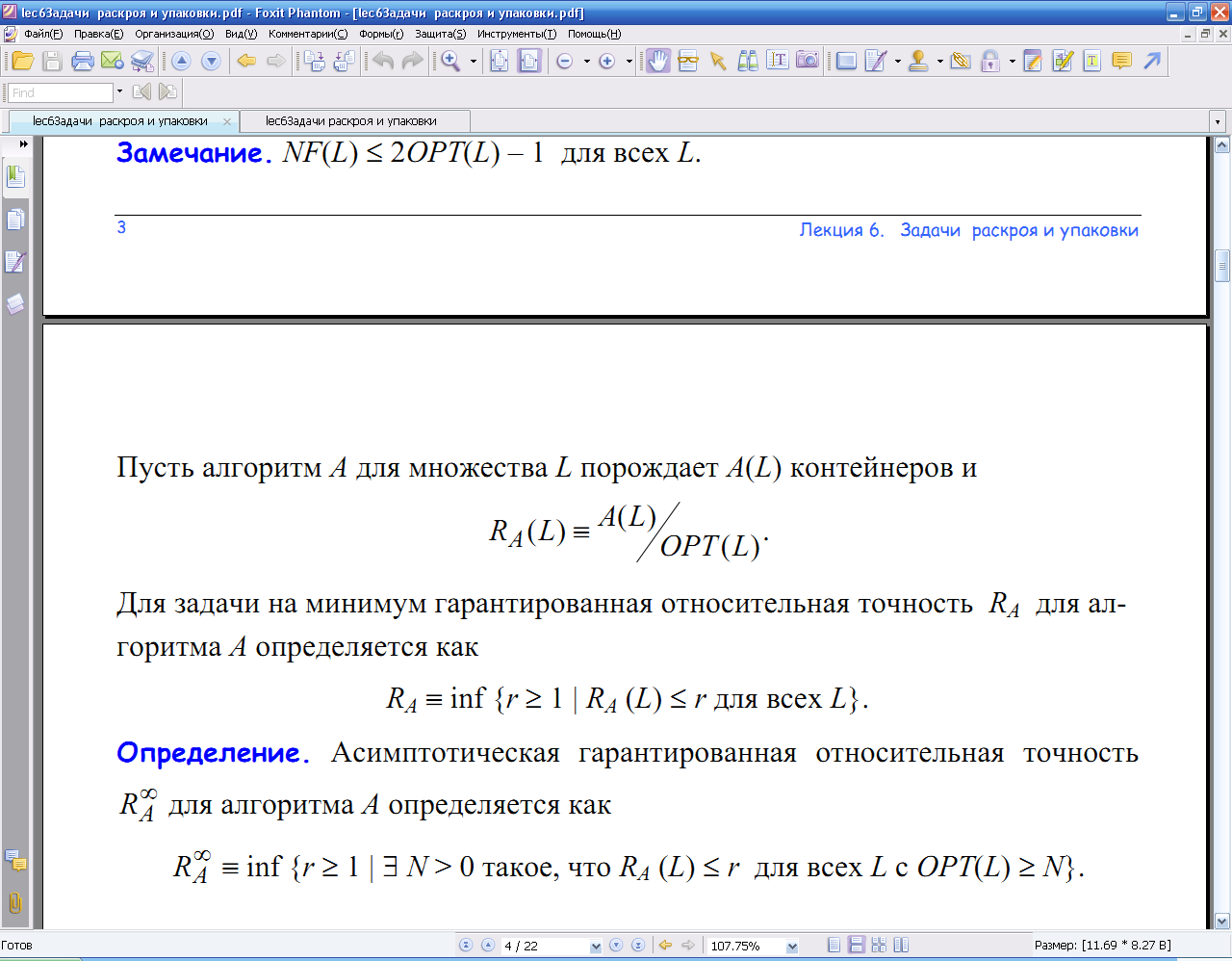
Метод SVC представляет собой многопроходную эвристику, предназначенную для решения задач одно-, двух- и трехмерного раскроя. Идея метода, основанная на использовании экономического смысла двойственных переменных, была предложена Э. А. Мухачевой и В. А. Залгаллером.

Метод SVC реализуется по схеме FFD с процедурами приоритета и повторения. Например, применяется простейшая схема FFD, она имитирует действие мастера в раскройном цехе. Пусть известен приоритетный список (Priority List, PL),в котором каким-то образом упорядочены заготовки. На j-м такте от стержня отрезается первая из PL возможныная заготовка с номером i длина  которой не превосходит длины L стержня. Потребность  в этой заготовке корректируют, полагая . Если новое значение , то элемент i удаляется из PL. От полученного остатка длины  вновь отрезается первая из PL возможная заготовка. Если таковой нет, то такт закончен и полученный раскрой j характеризуется вектором , компоненты  которого указывают количество i-х заготовок в j-м раскрое. Следующий (j+1)-й такт начинается с раскроя целого стержня. Если PL=Ø, т.е. все требуемые заготовки получены, то конец. В методе SVC упорядочивание элементов основано на экономическом смысле объективно-обусловленных оценок (о. о. оценок) Л. В. Канторовича в линейном программировании. А именно, они имеют смысл подетальных норм расхода материала в оптимальном плане. В допустимом плане линейного непрерывного раскроя в качестве оценок принимают двойственные переменные. С помощью последних проверяется оптимальность плана и в противном случае намечается путь его улучшения. В целочисленном случае имеется разрыв двойственности и непосредственное применение линейного программирования и о. о. оценок становится невозможным. С целью улучшения плана раскроя используют только идею пересчета оценок

2.4 Алгоритмы типа On-line

Предметы поступают в непредсказуемом порядке. Требуется упаковать их в минимальное число контейнеров. Упакованный предмет нельзя перемещать в другой контейнер. Место для предварительного хранения предметов отсутствует.

Алгоритмы NF, FF, BF являются On-line алгоритмами.

Теорема. Для любого On-line алгоритма A справедливо неравенство  >1.5

On-line алгоритм называют алгоритмом с ограниченным доступом к контейнерам, если на каждом шаге алгоритм имеет возможность помещать предметы только в один из K контейнеров (K — const). Эти контейнеры называются открытыми. Если контейнер закрыли, то он уже не открывается (например, отправляется потребителю). Прежде чем добавить пустой открытый контейнер, нужно закрыть один из K открытых контейнеров.

Алгоритм NF — пример для K = 1.

Правила для выбора контейнера

1. Закрыть контейнер с наименьшим номером

2. Закрыть самый заполненный контейнер.

Примеры алгоритмов с ограниченным доступом

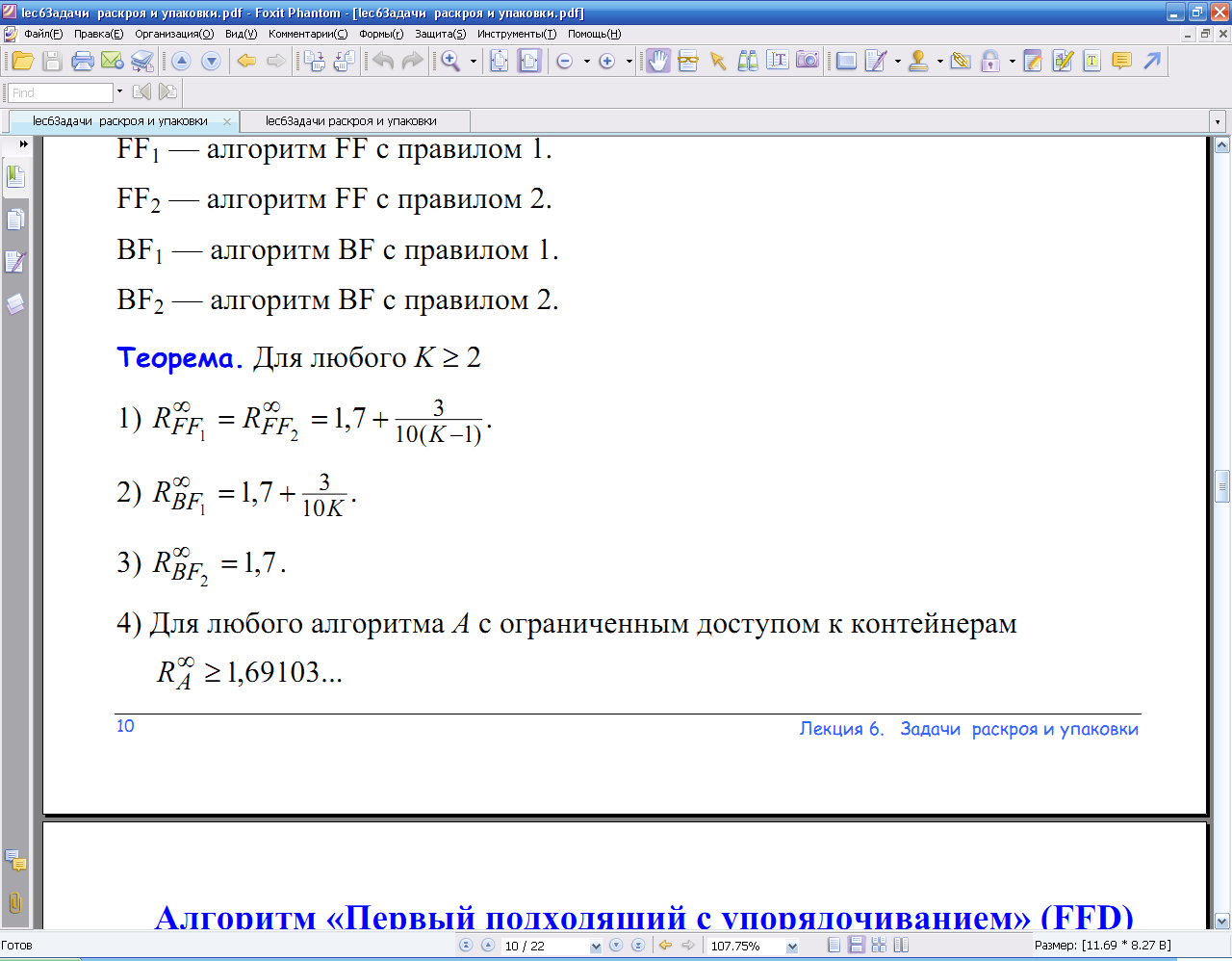
FF1 — алгоритм FF с правилом 1.

FF2 — алгоритм FF с правилом 2.

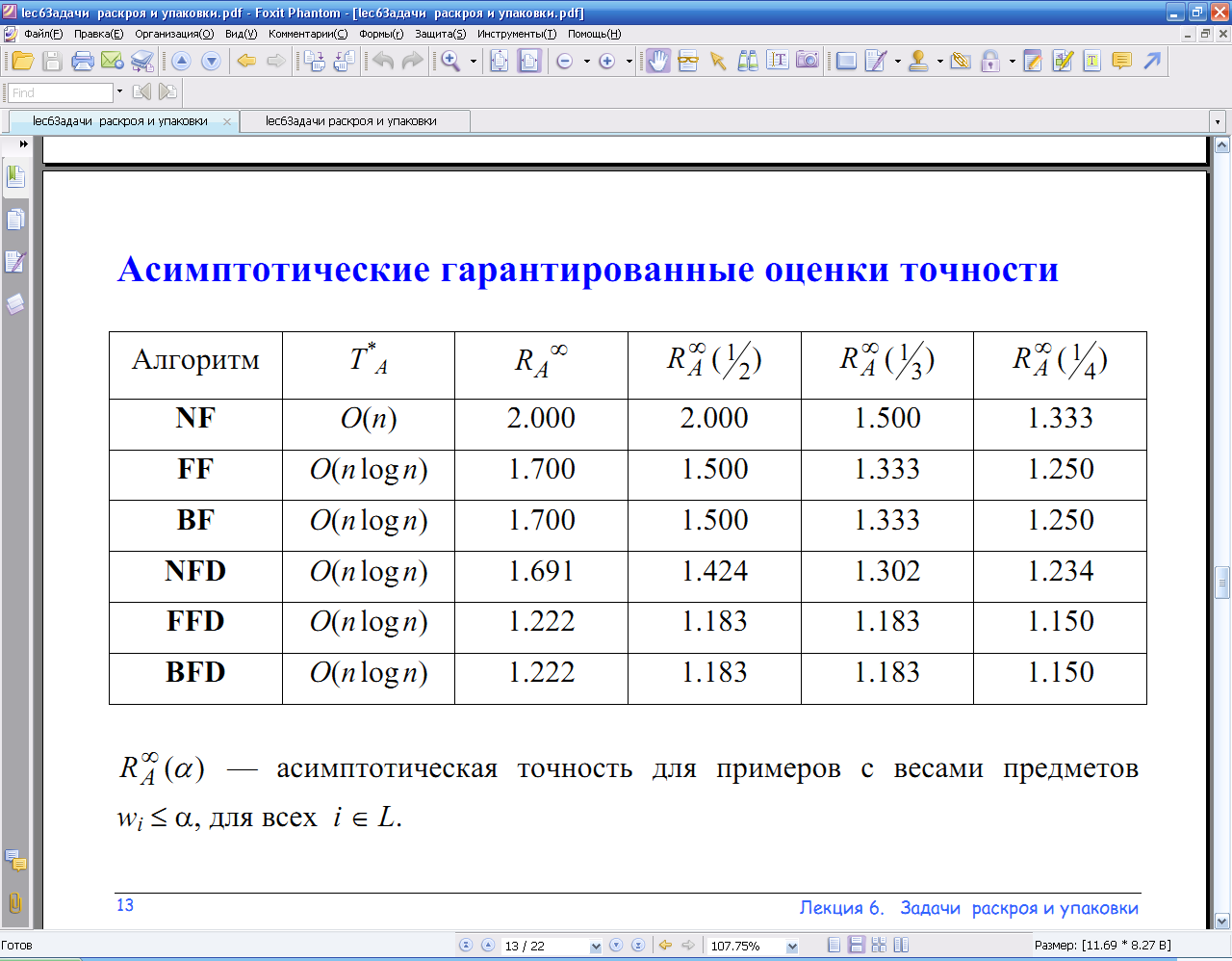
BF1 — алгоритм BF с правилом 1.

BF2 — алгоритм BF с правилом 2.

Теорема. Для любого K ≥ 2



Асимптотические гарантированные оценки точности



2.4.1 Алгоритм «Следующий подходящий» (NF)

В произвольном порядке упаковываем предметы по следующему правилу.

Первый предмет помещаем в первый контейнер.

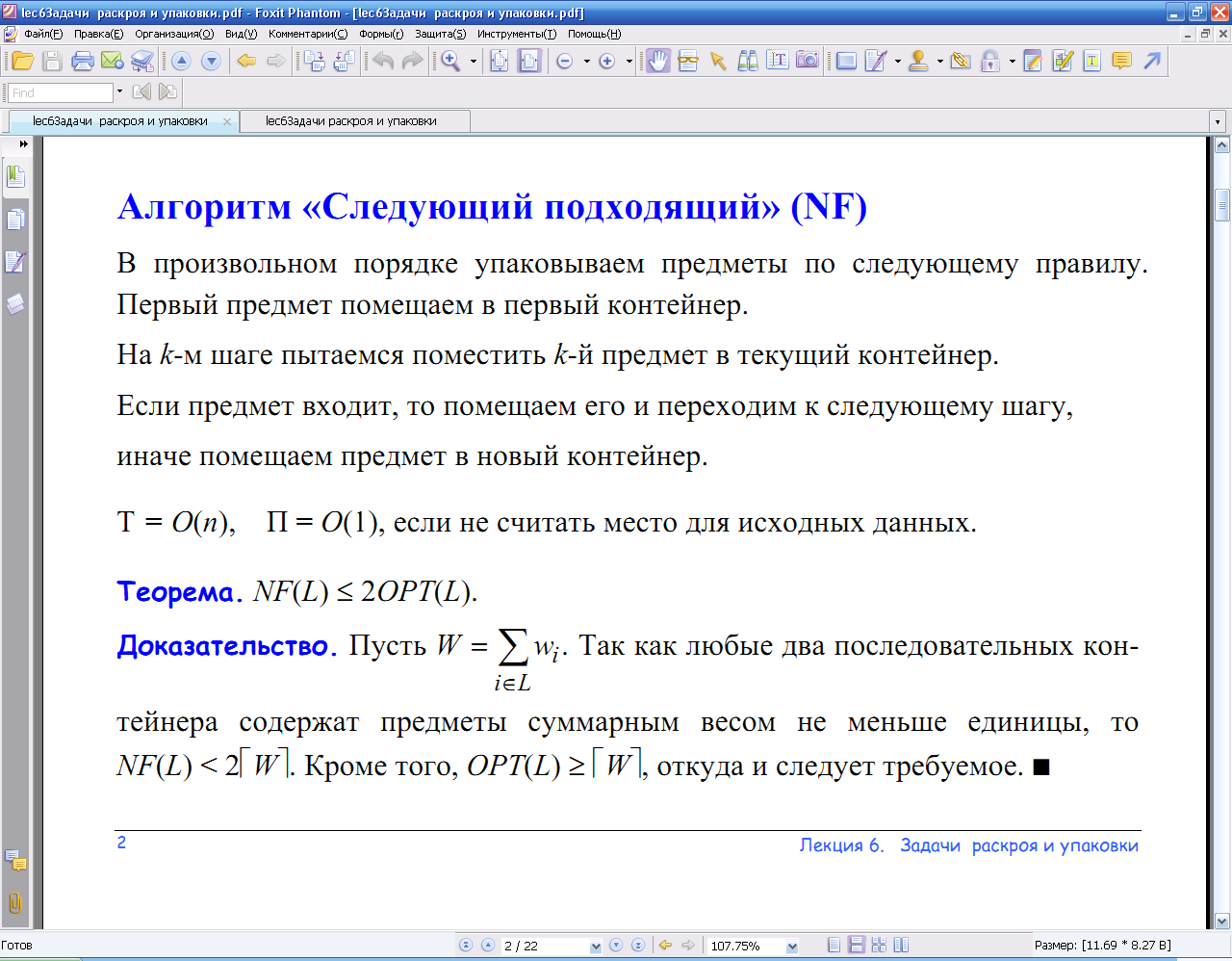
На k-м шаге пытаемся поместить k-й предмет в текущий контейнер.

Если предмет входит, то помещаем его и переходим к следующему шагу, иначе помещаем предмет в новый контейнер.

Т = О(n), П = О(1), если не считать место для исходных данных.

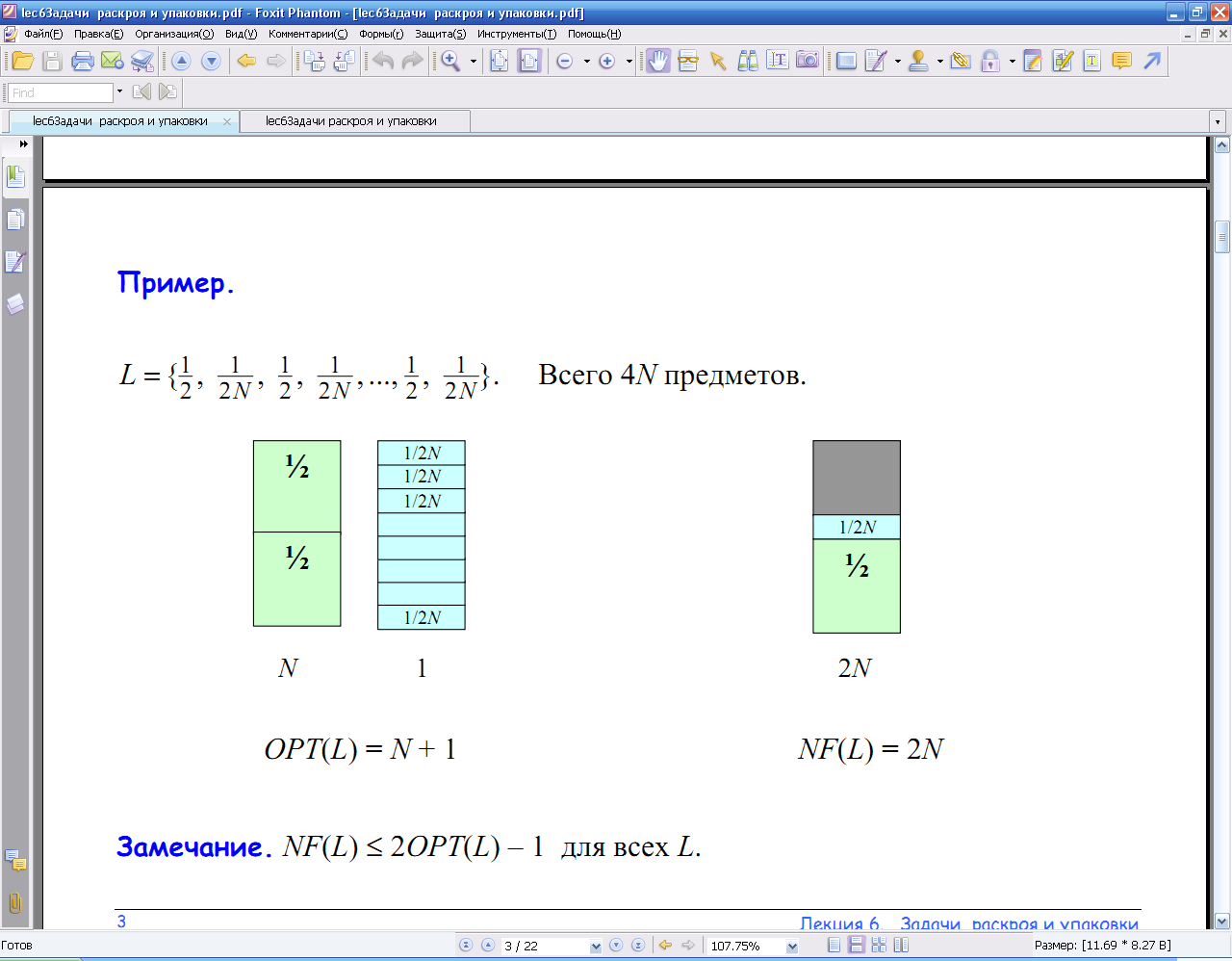
Теорема. NF(L) ≤ 2OPT(L).

Доказательство. Пусть



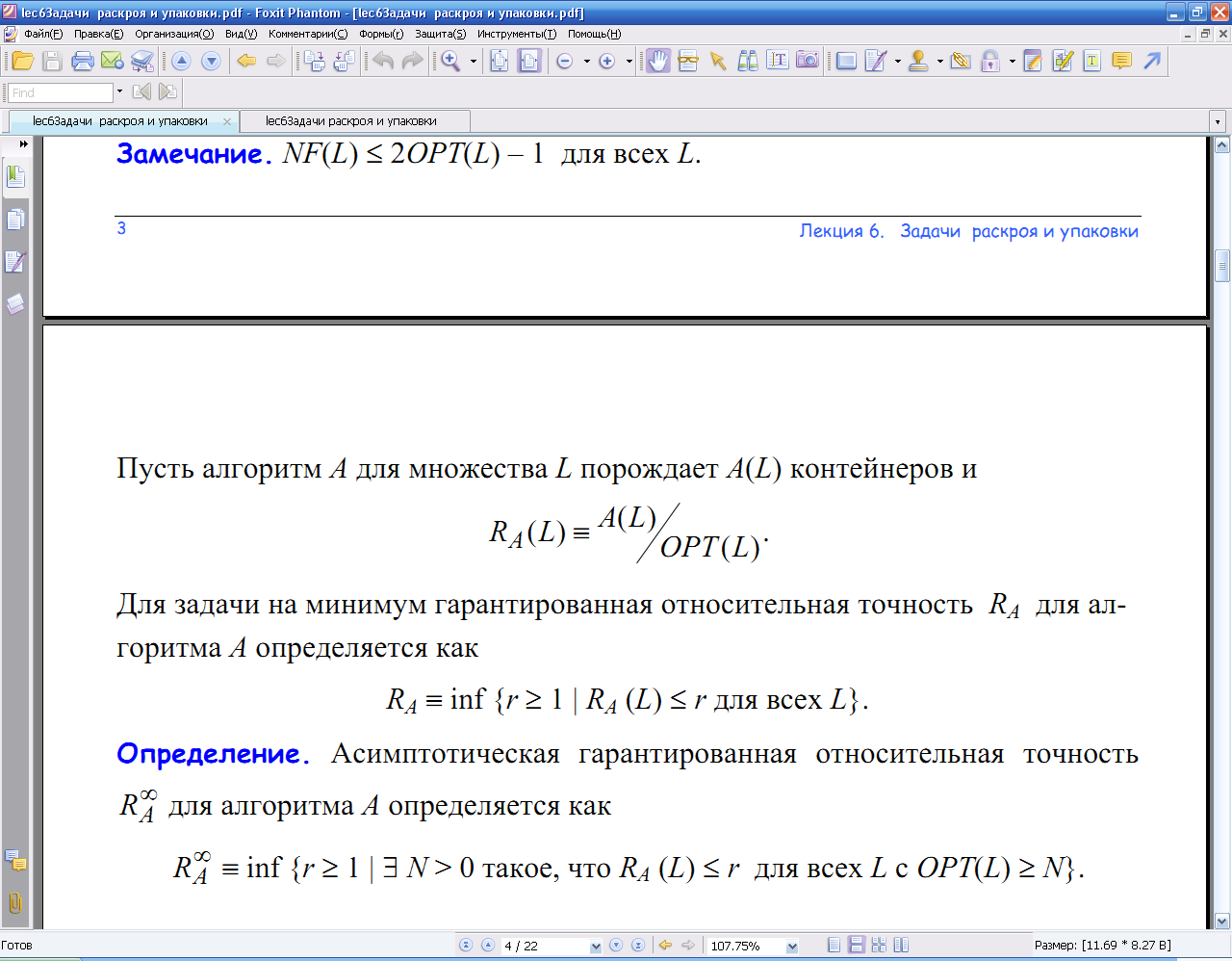
Так как любые два последовательных контейнера содержат предметы суммарным весом не меньше единицы, то NF(L) < 2[W]. Кроме того, OPT(L) ≥ [W], откуда и следует требуемое.

Пример.



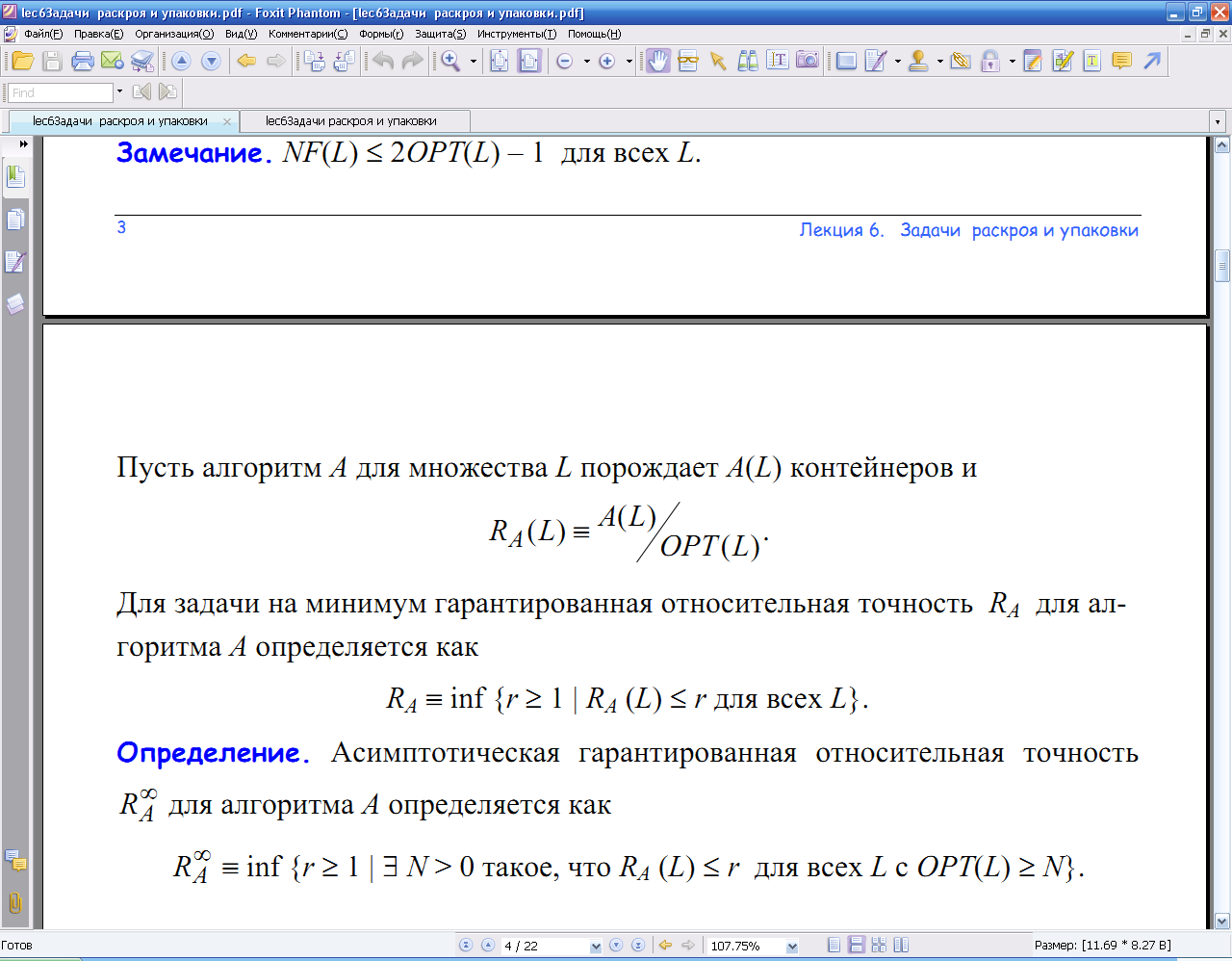
Замечание. NF(L) ≤ 2OPT(L) – 1 для всех L.

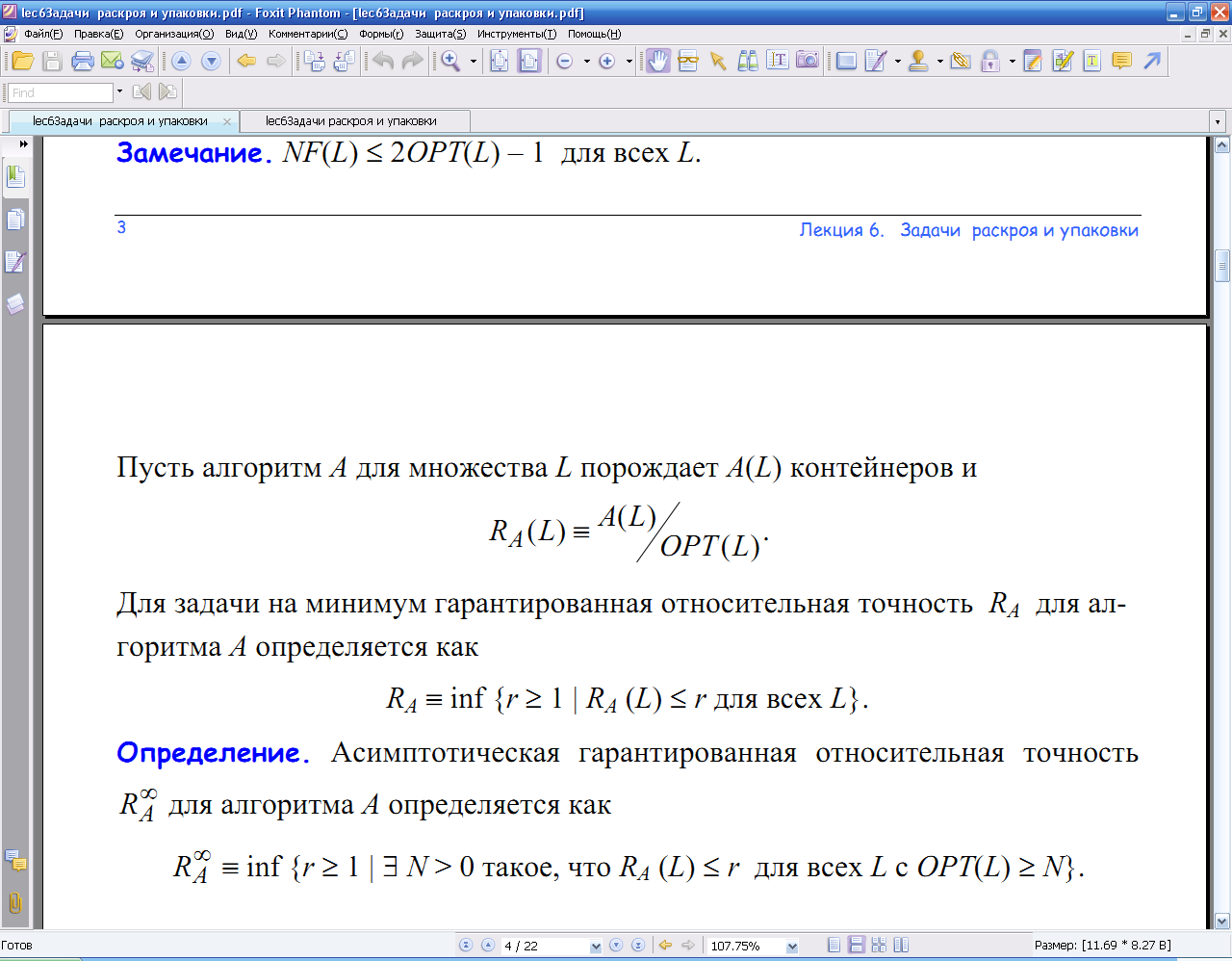
Пусть алгоритм A для множества L порождает A(L) контейнеров и



Для задачи на минимум гарантированная относительная точность RA для алгоритма A определяется как

RA ≡ inf {r ≥ 1 | RA (L) ≤ r для всех L}.

Определение. Асимптотическая гарантированная относительная точность для алгоритма A определяется как

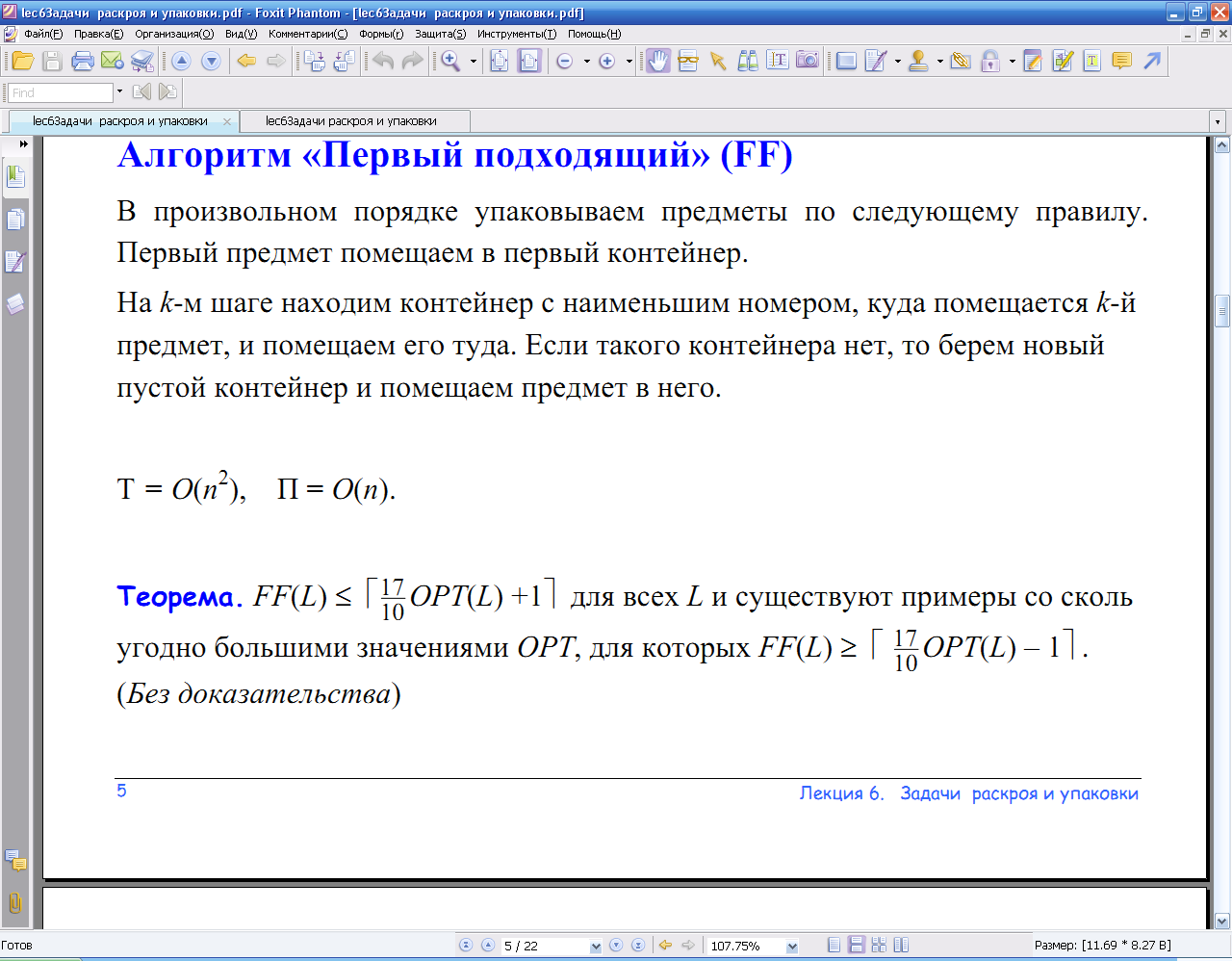
≡ inf {r ≥ 1 | ∃ N > 0 такое, что RA (L) ≤ r для всех L с OPT(L) ≥ N}.

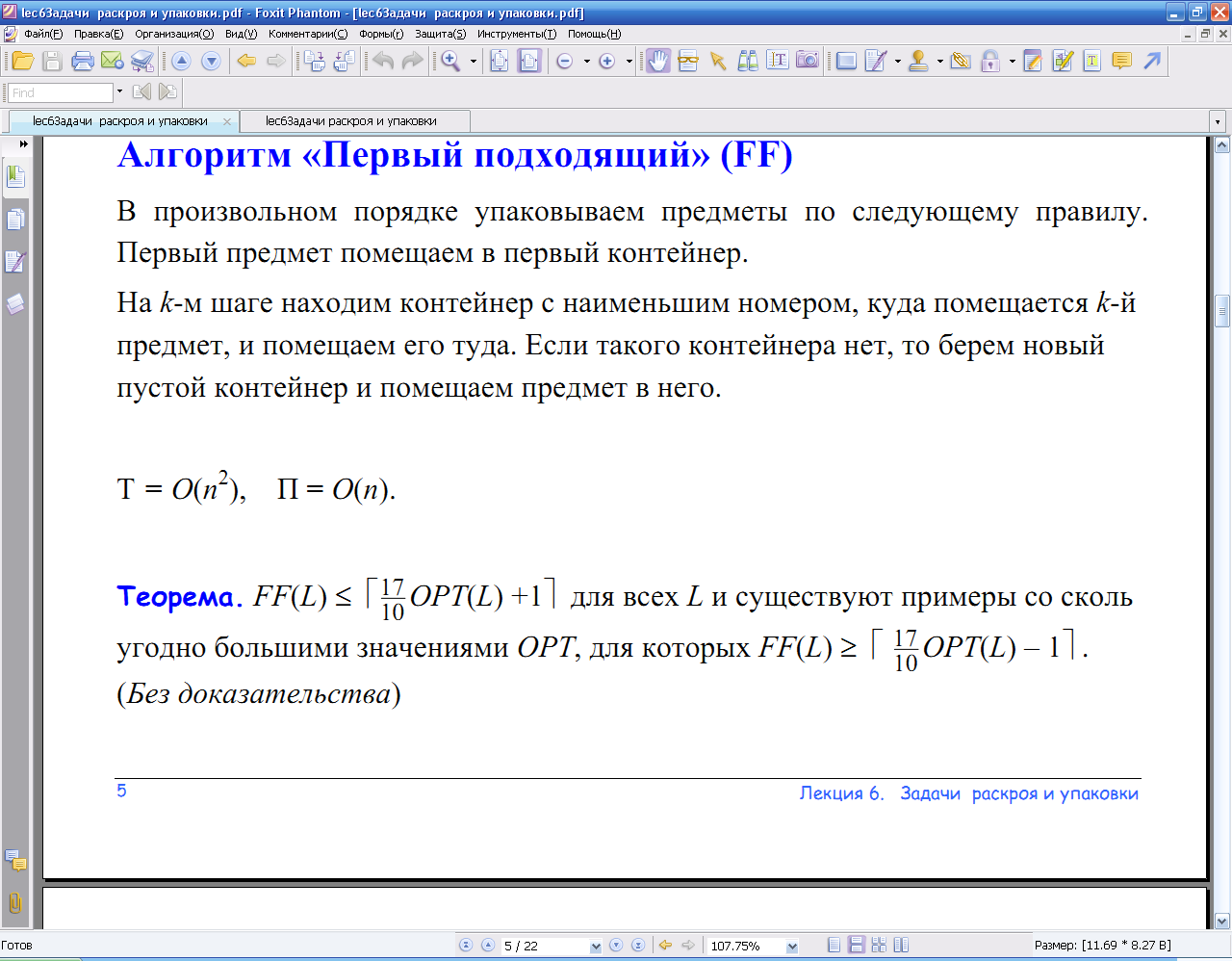
2.4.2 Алгоритм «Первый подходящий» (FF)

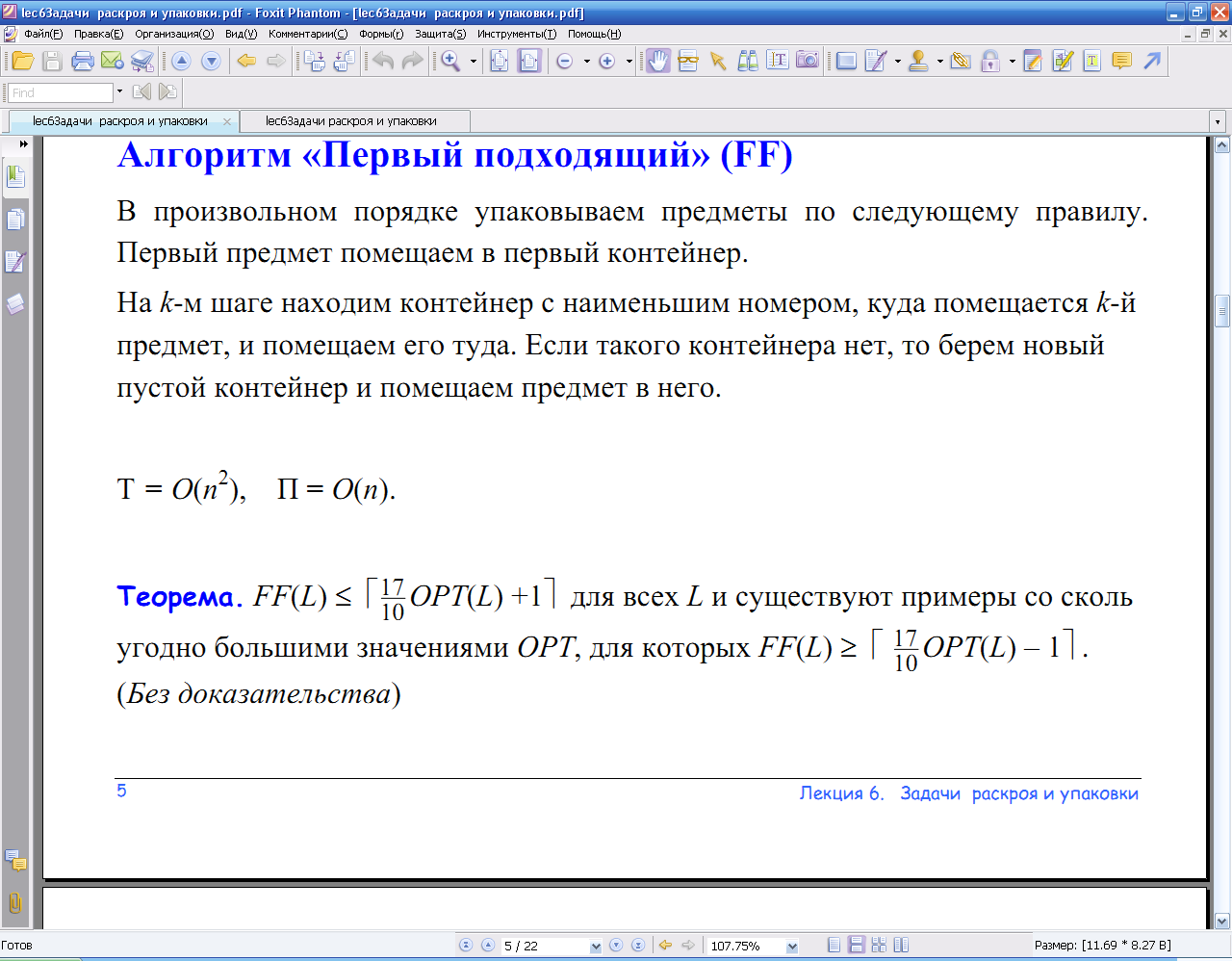
В произвольном порядке упаковываем предметы по следующему правилу.

Первый предмет помещаем в первый контейнер.

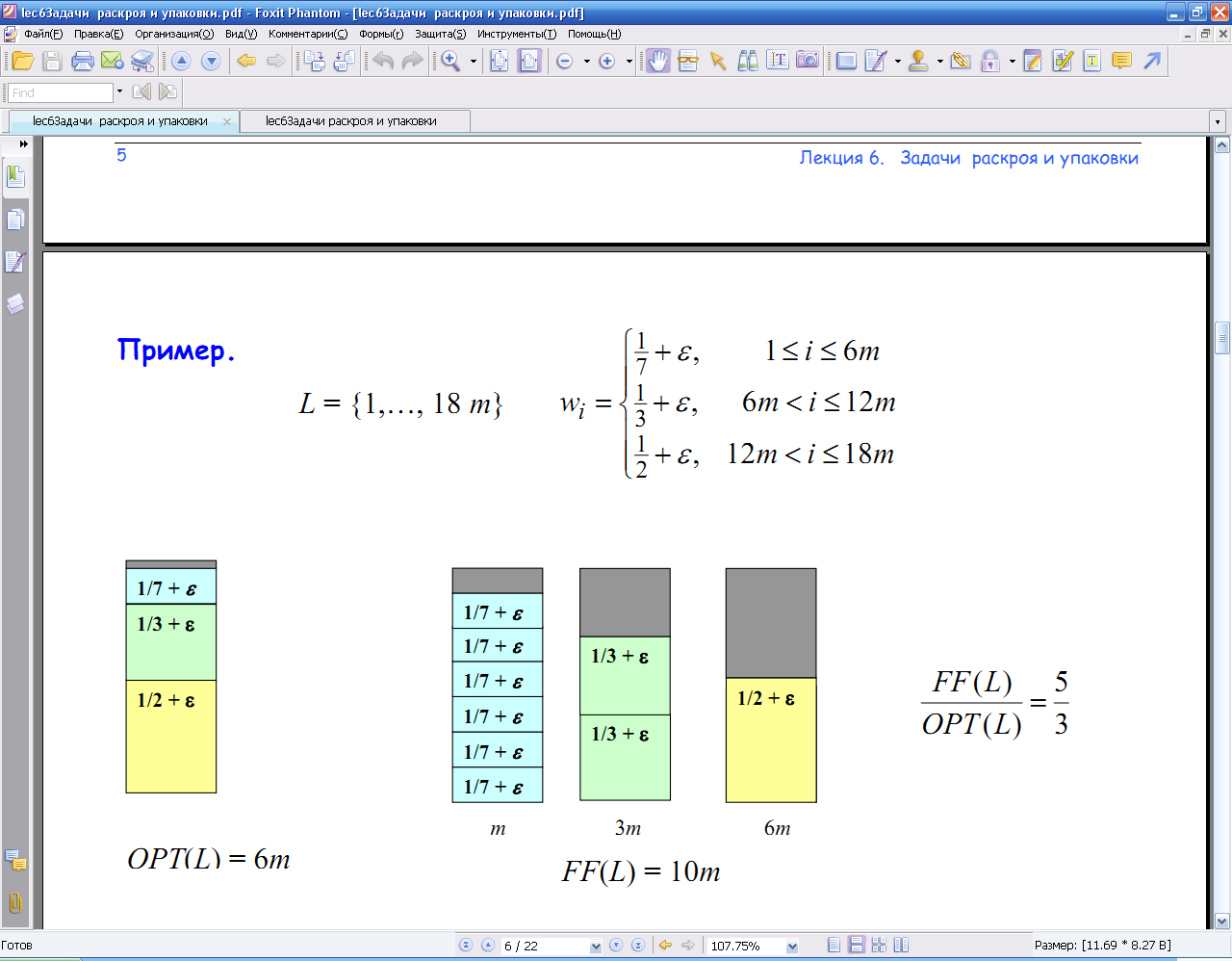
На k-м шаге находим контейнер с наименьшим номером, куда помещается k-й предмет, и помещаем его туда. Если такого контейнера нет, то берем новый пустой контейнер и помещаем предмет в него.

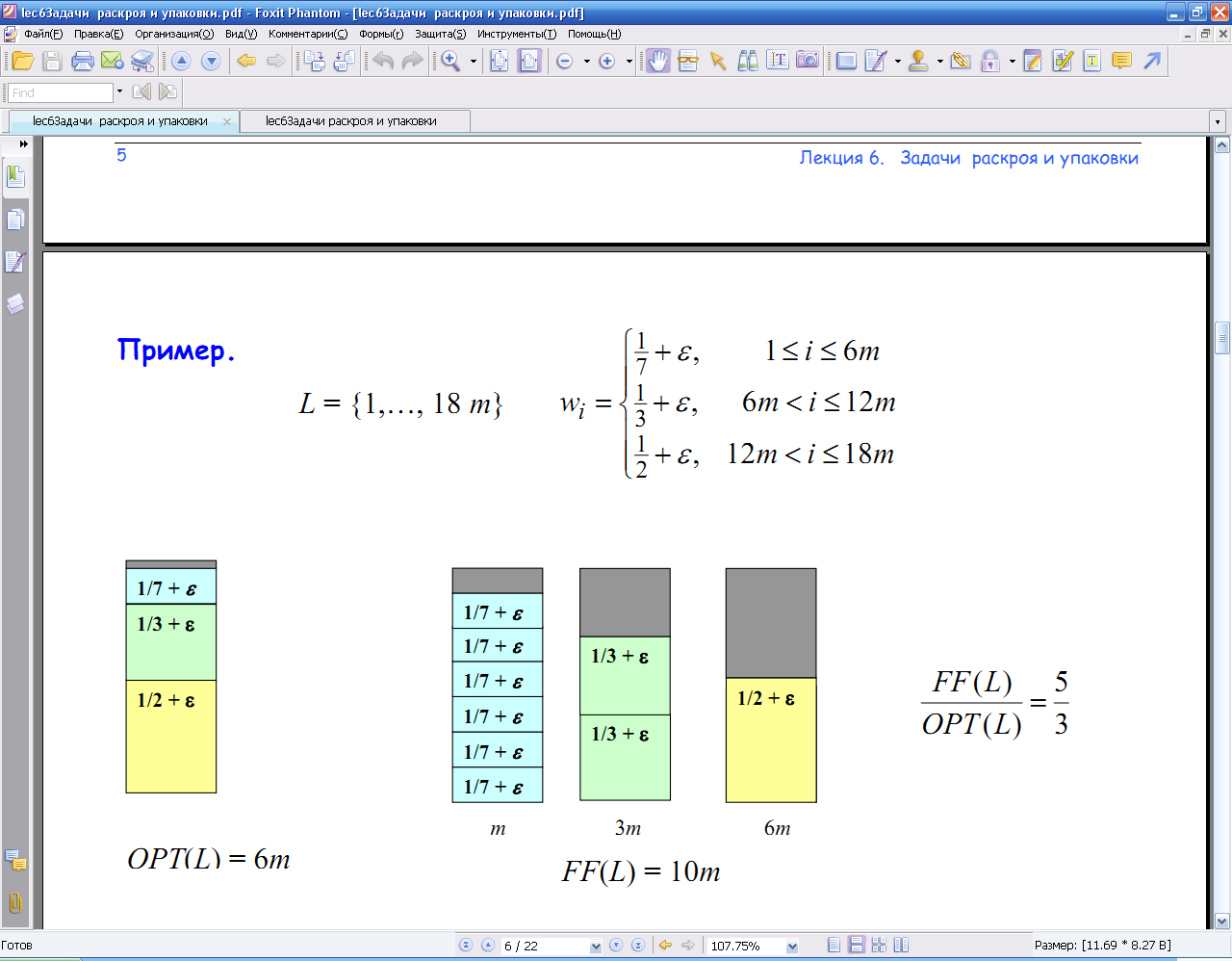


Теорема. для всех L и существуют примеры со сколь

угодно большими значениями OPT, для которых 

Пример.

⎪

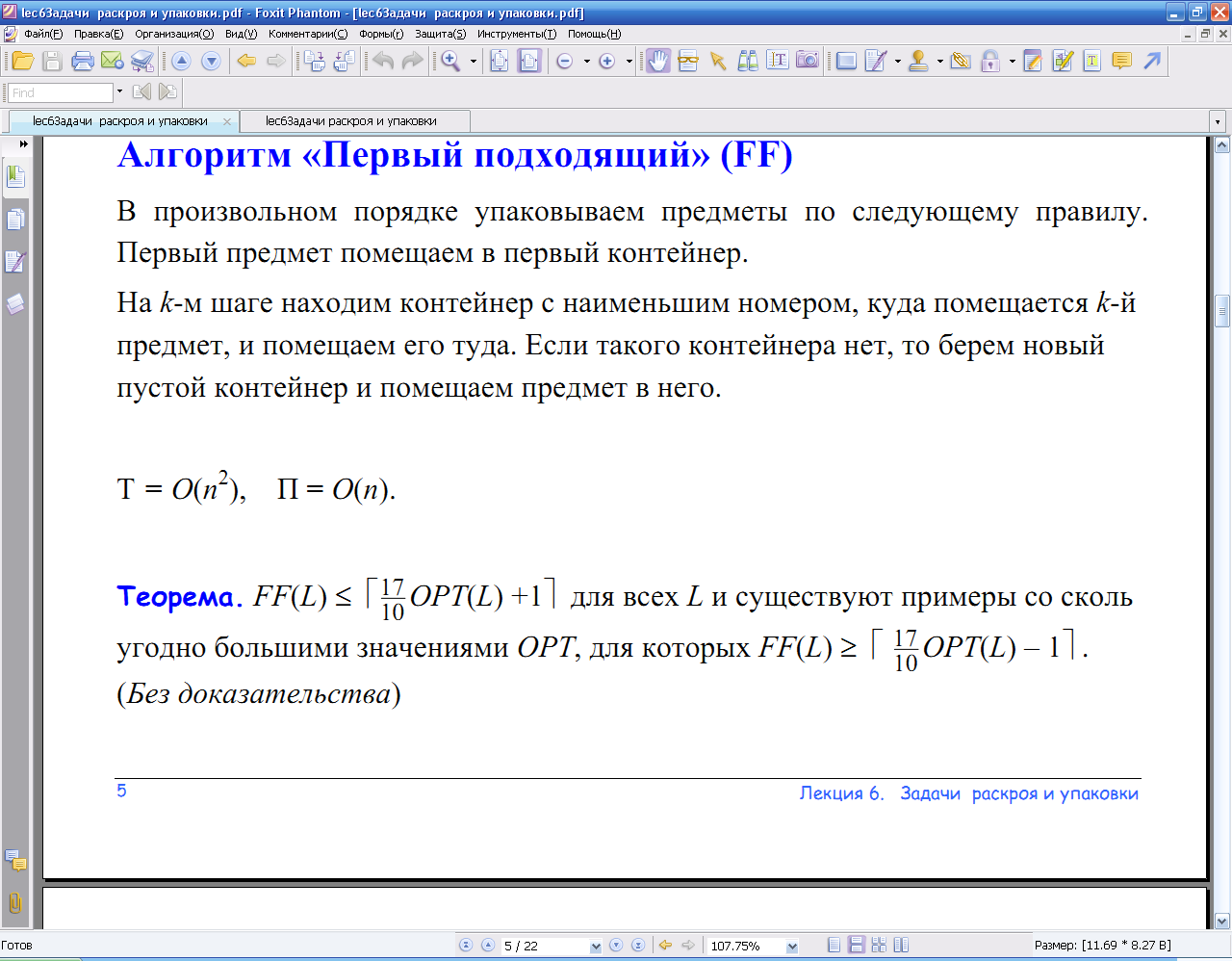


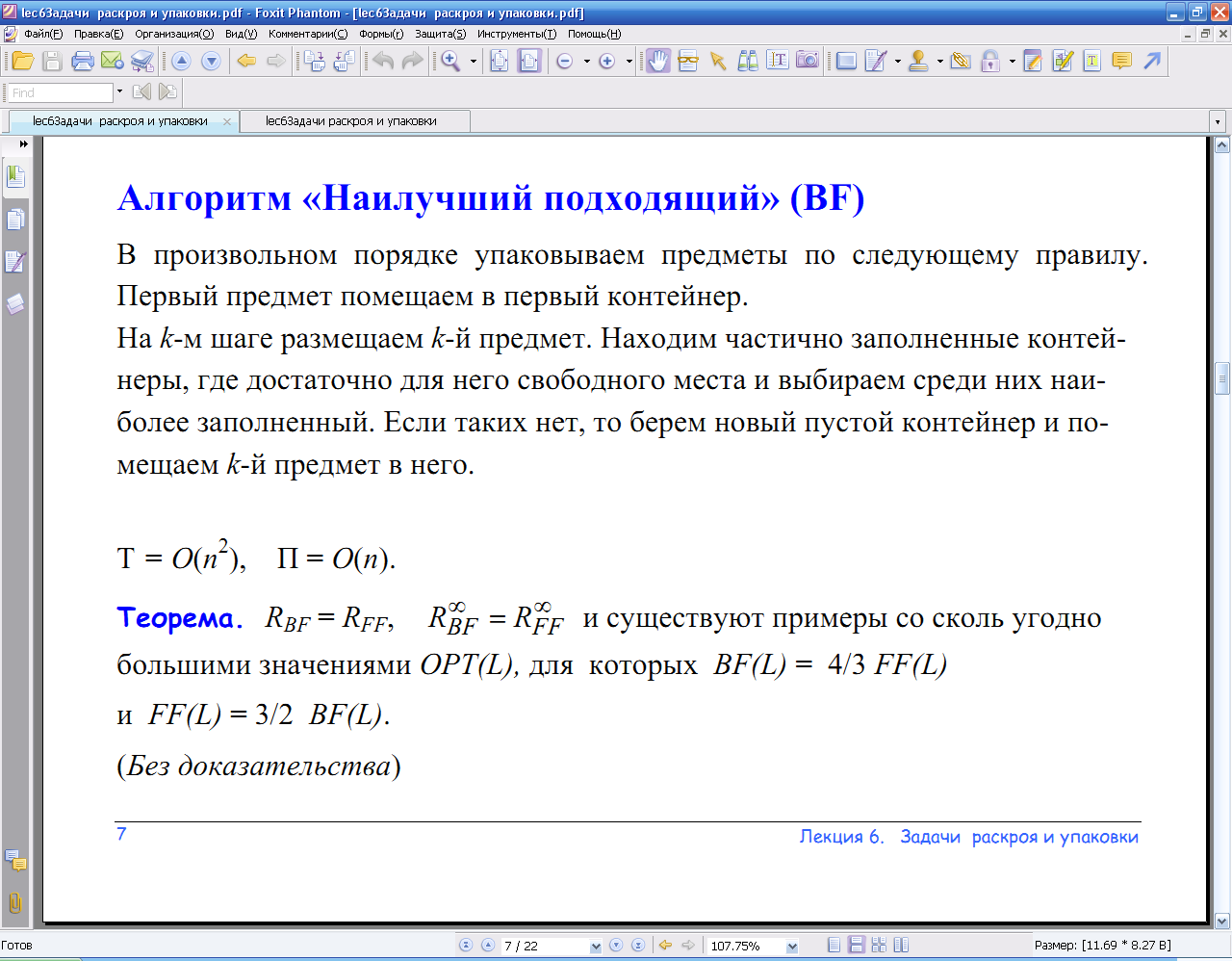
2.4.3 Алгоритм «Наилучший подходящий» (BF)

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.



Теорема. и существуют примеры со сколь угодно

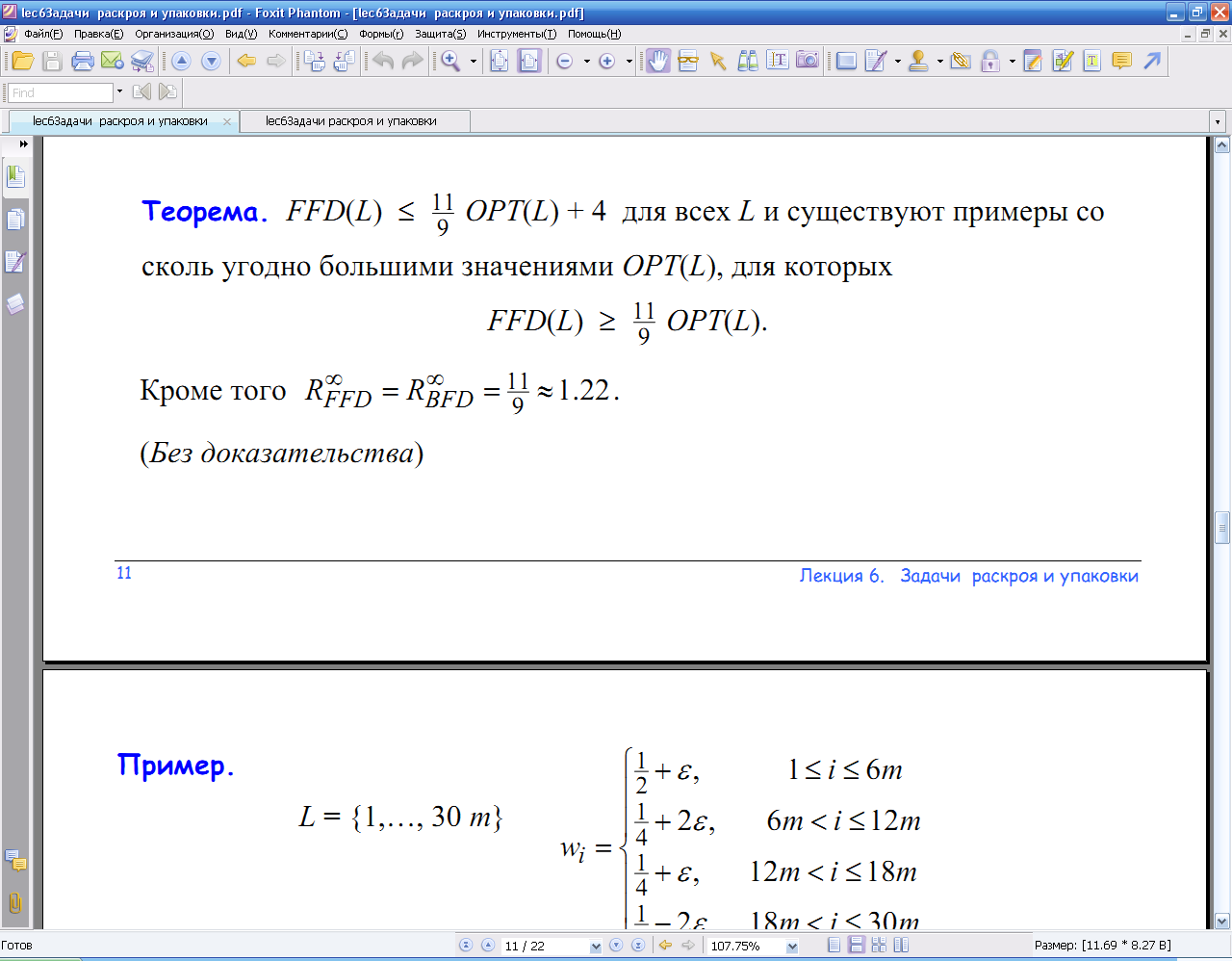
большими значениями OPT(L), для которых BF(L) = 4/3 FF(L) и FF(L) = 3/2 BF(L).

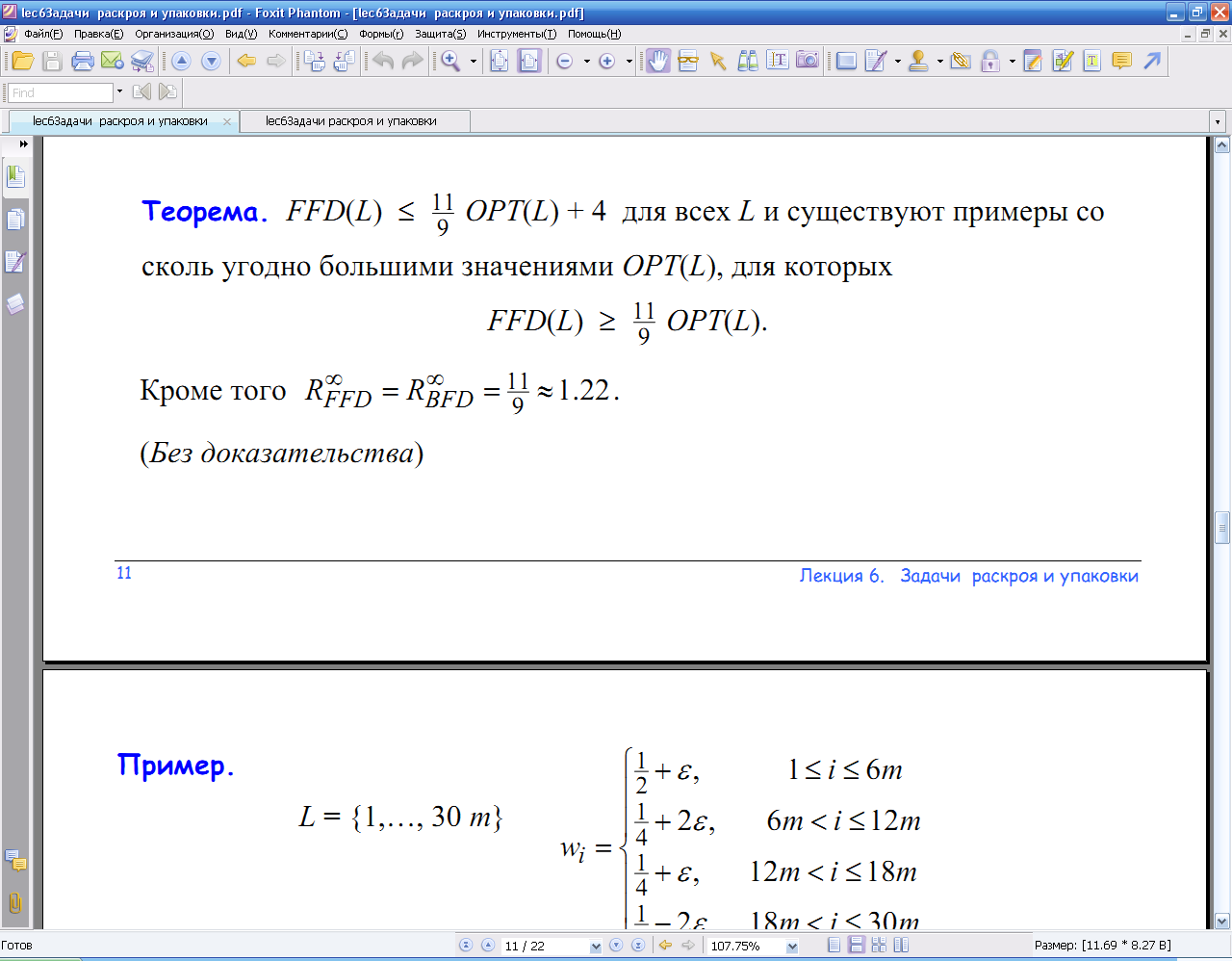
2.4.4 Алгоритм «Первый подходящий с упорядочиванием» (FFD)

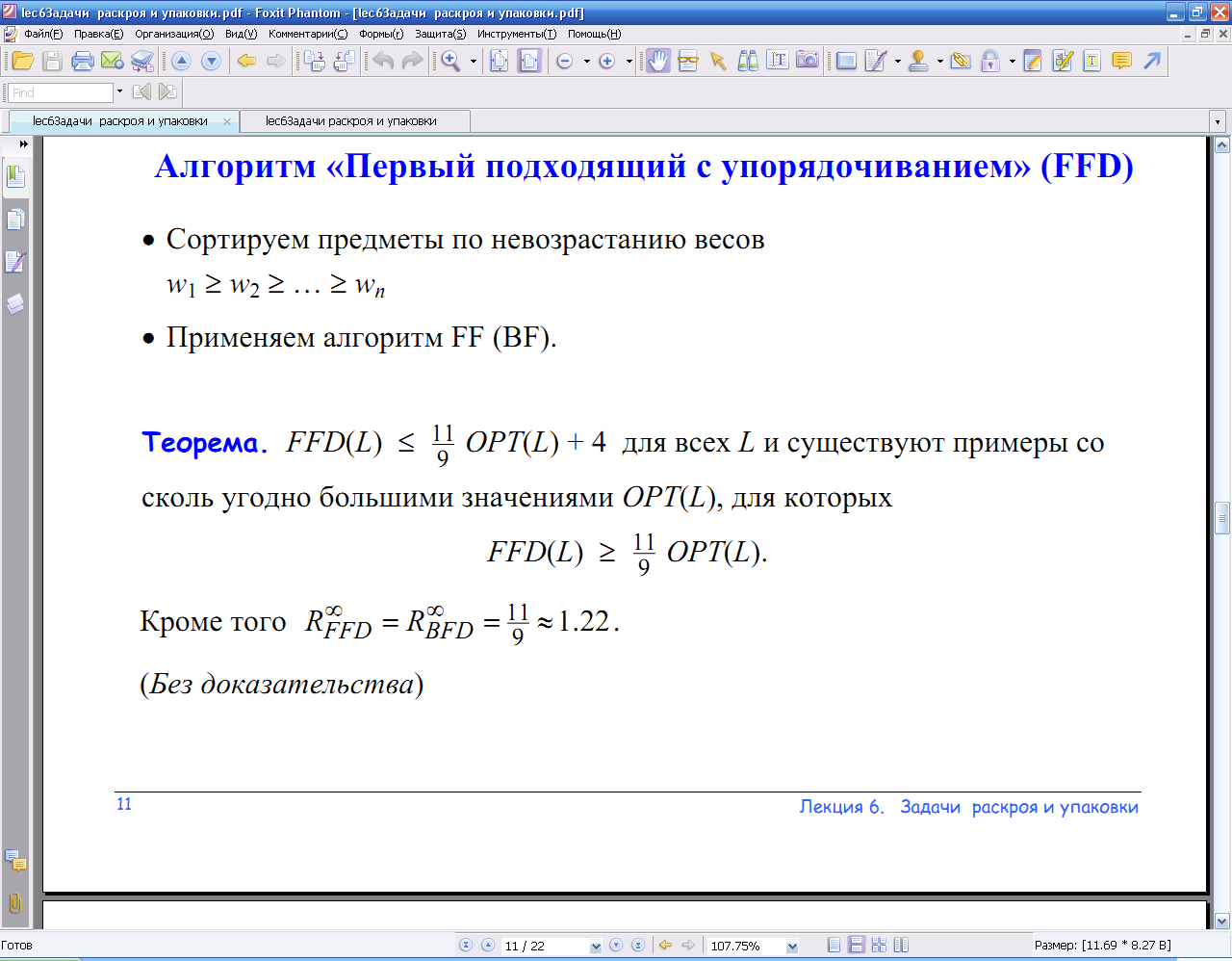
• Сортируем предметы по невозрастанию весов

w1 ≥ w2 ≥ … ≥ wn

• Применяем алгоритм FF (BF).

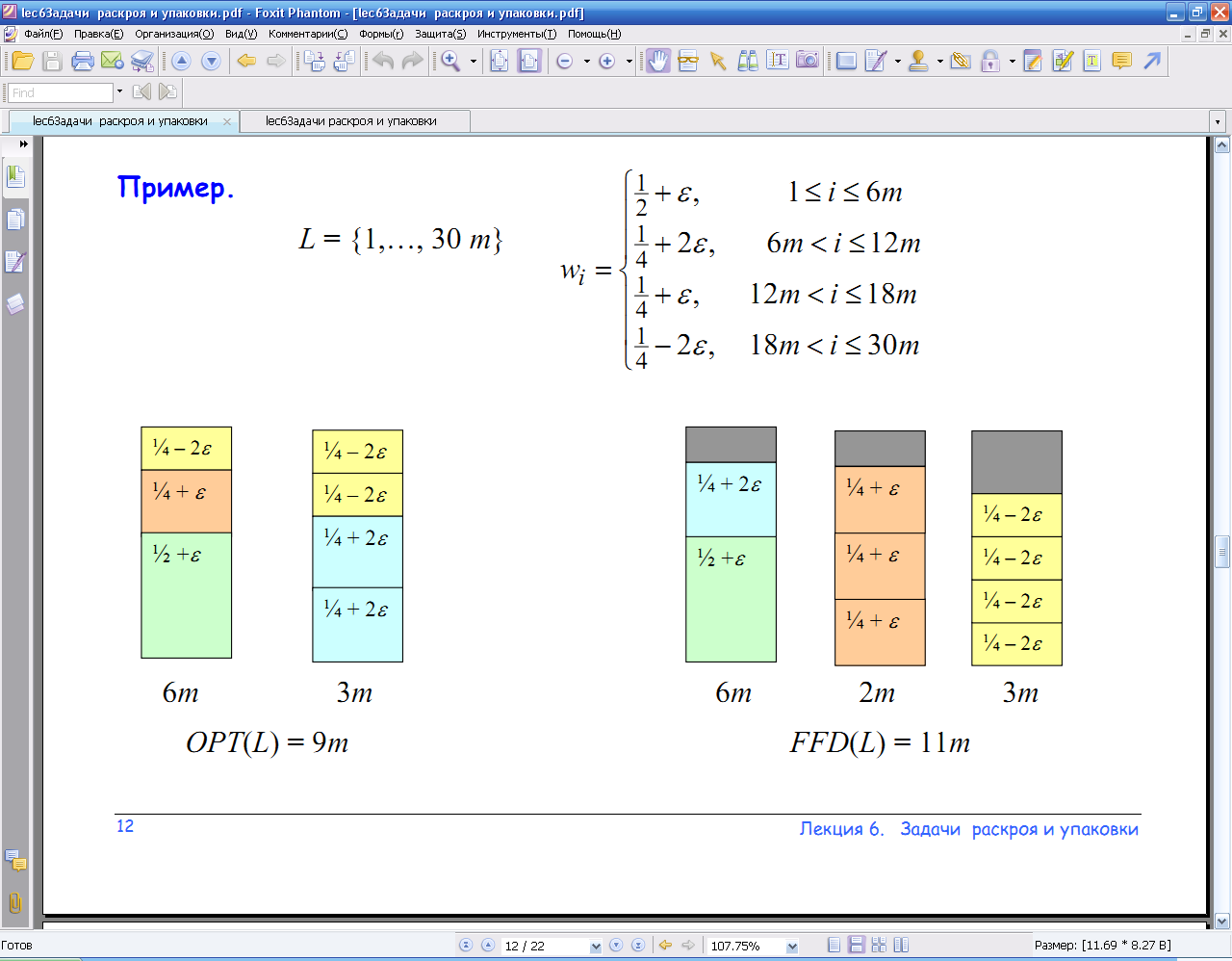
Теорема. FFD(L) ≤ OPT(L) + 4 для всех L и существуют примеры со сколь угодно большими значениями OPT(L), для которых

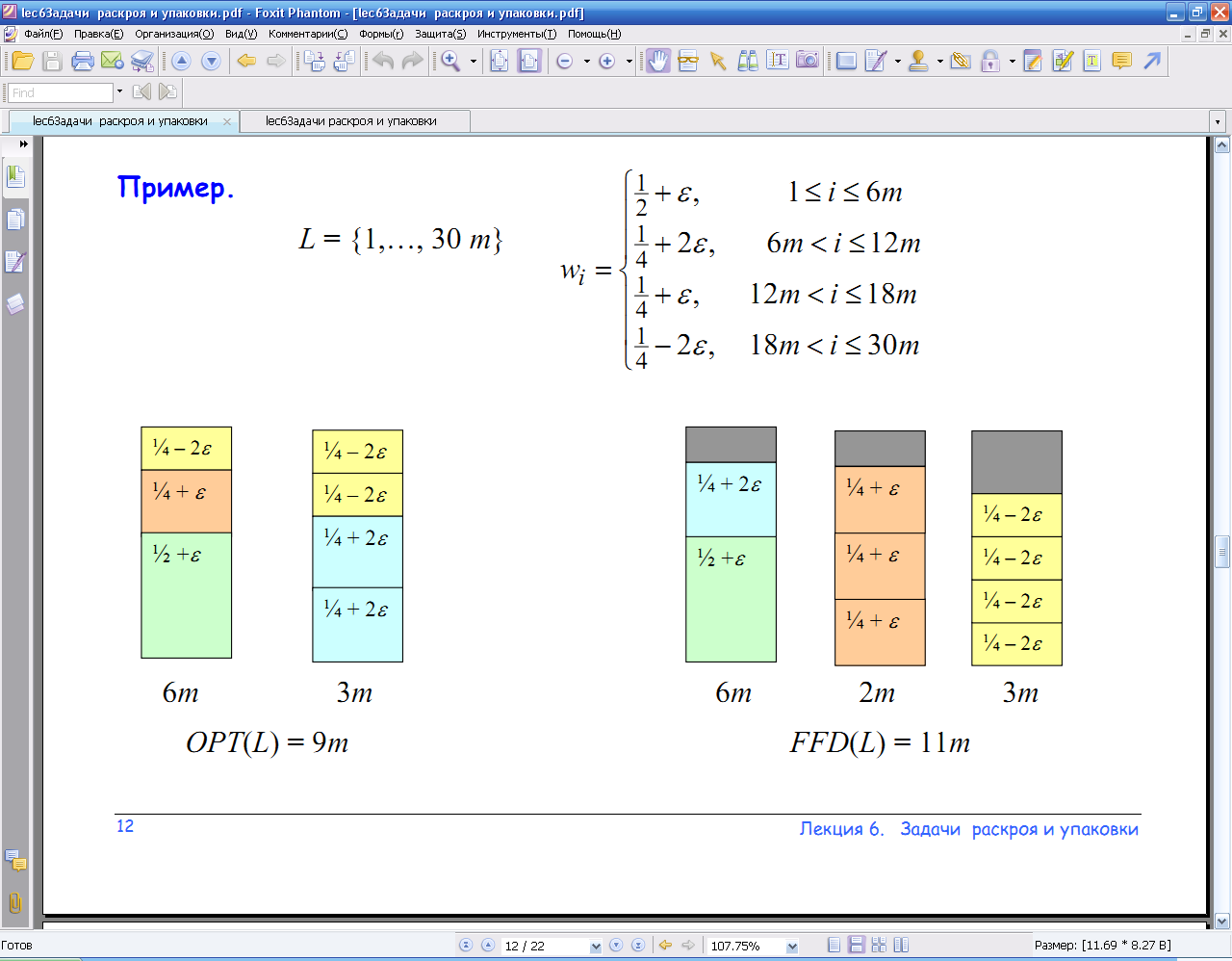
FFD(L) ≥  OPT(L).

Кроме того 

(Без доказательства)

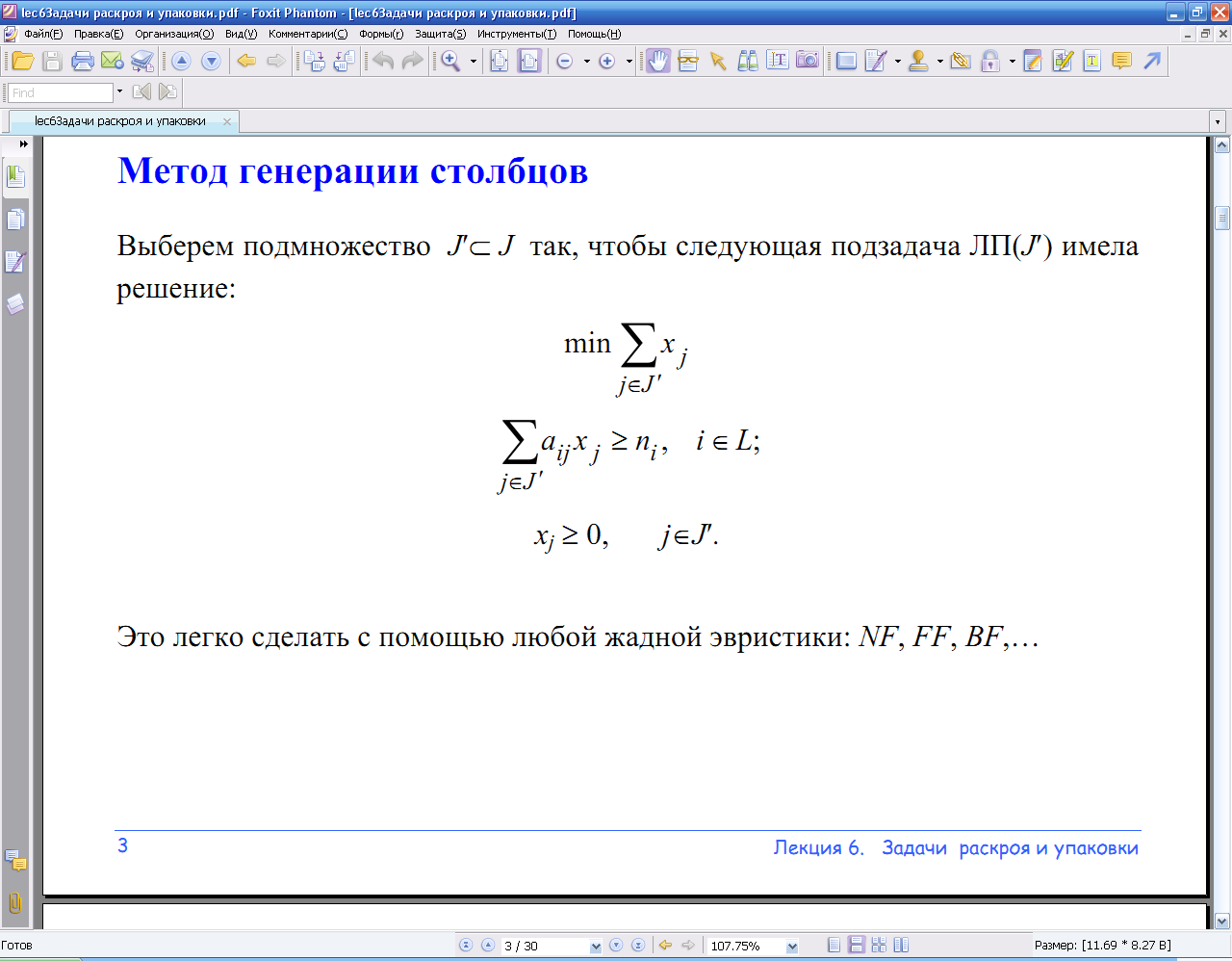
Пример.



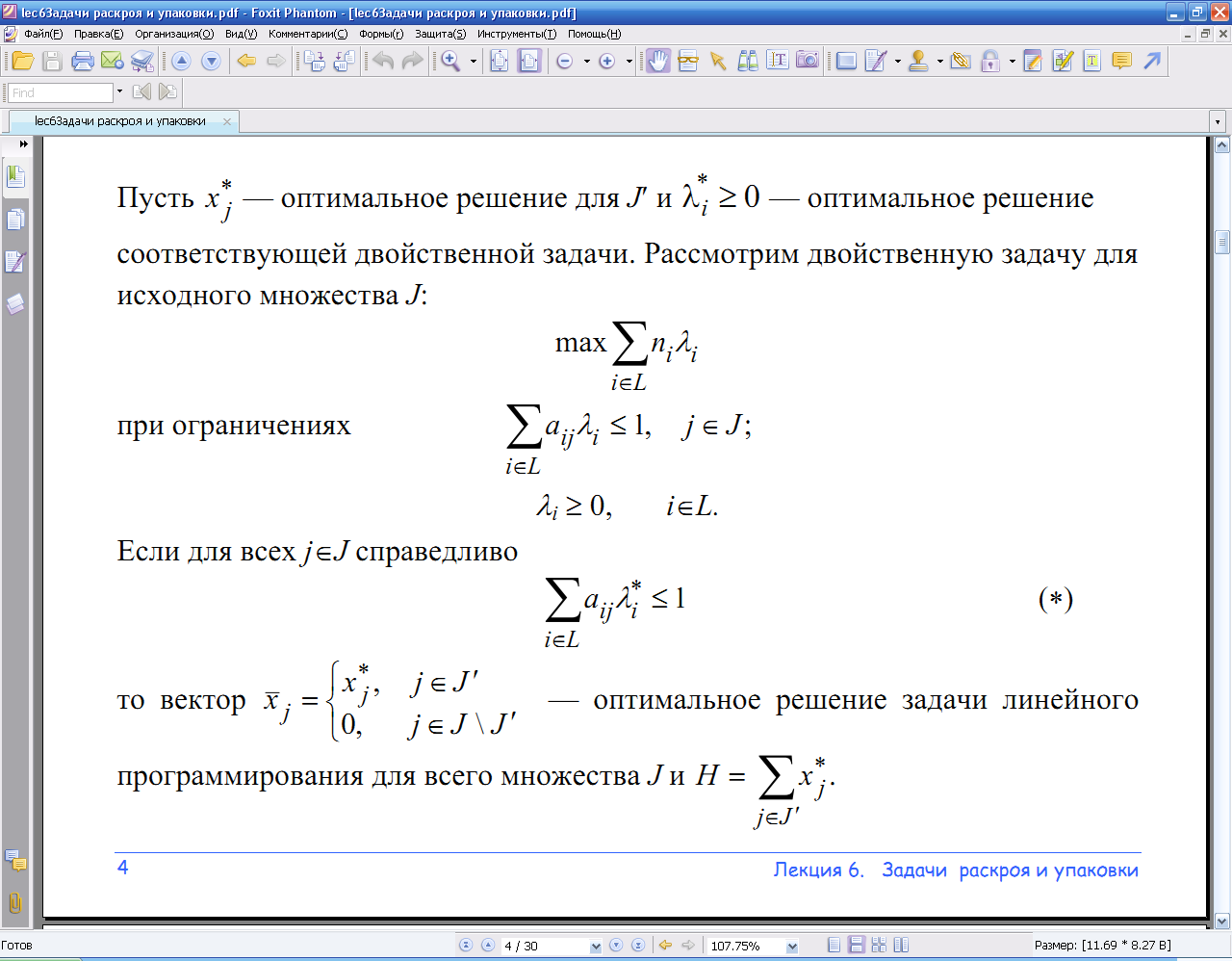
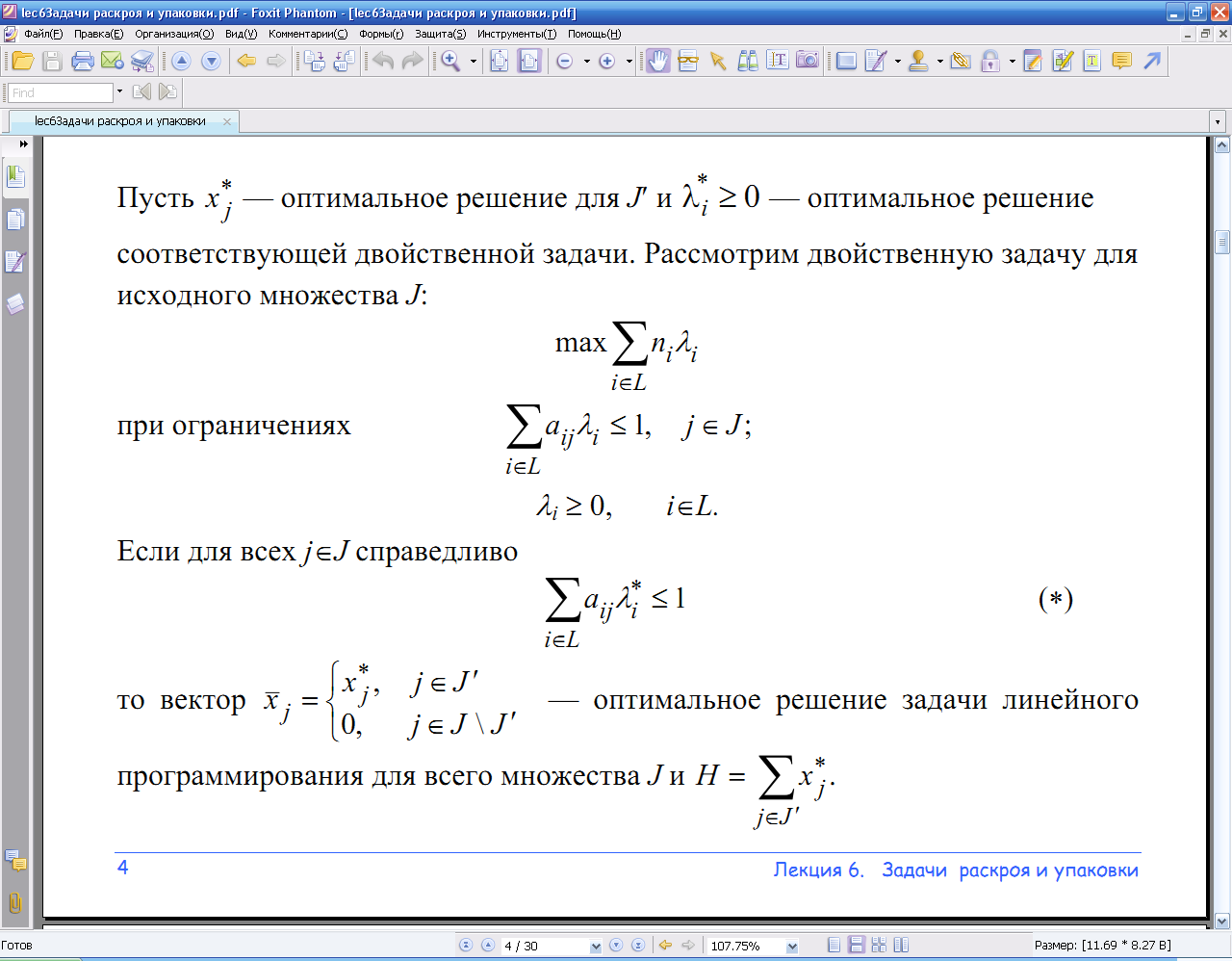


2.5 Метод генерации столбцов

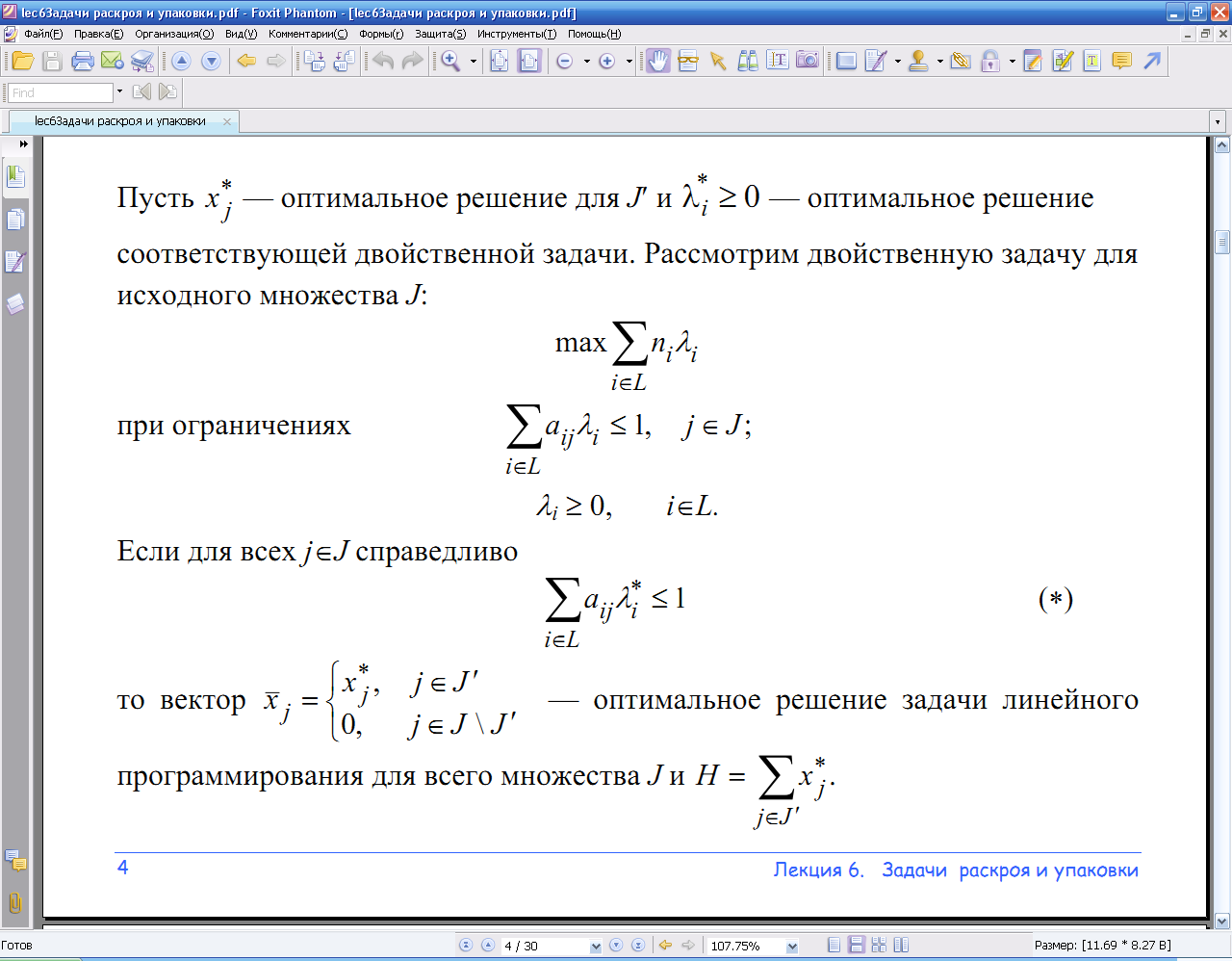
Выберем подмножество J′⊂ J так, чтобы следующая подзадача ЛП(J′) имела решение:



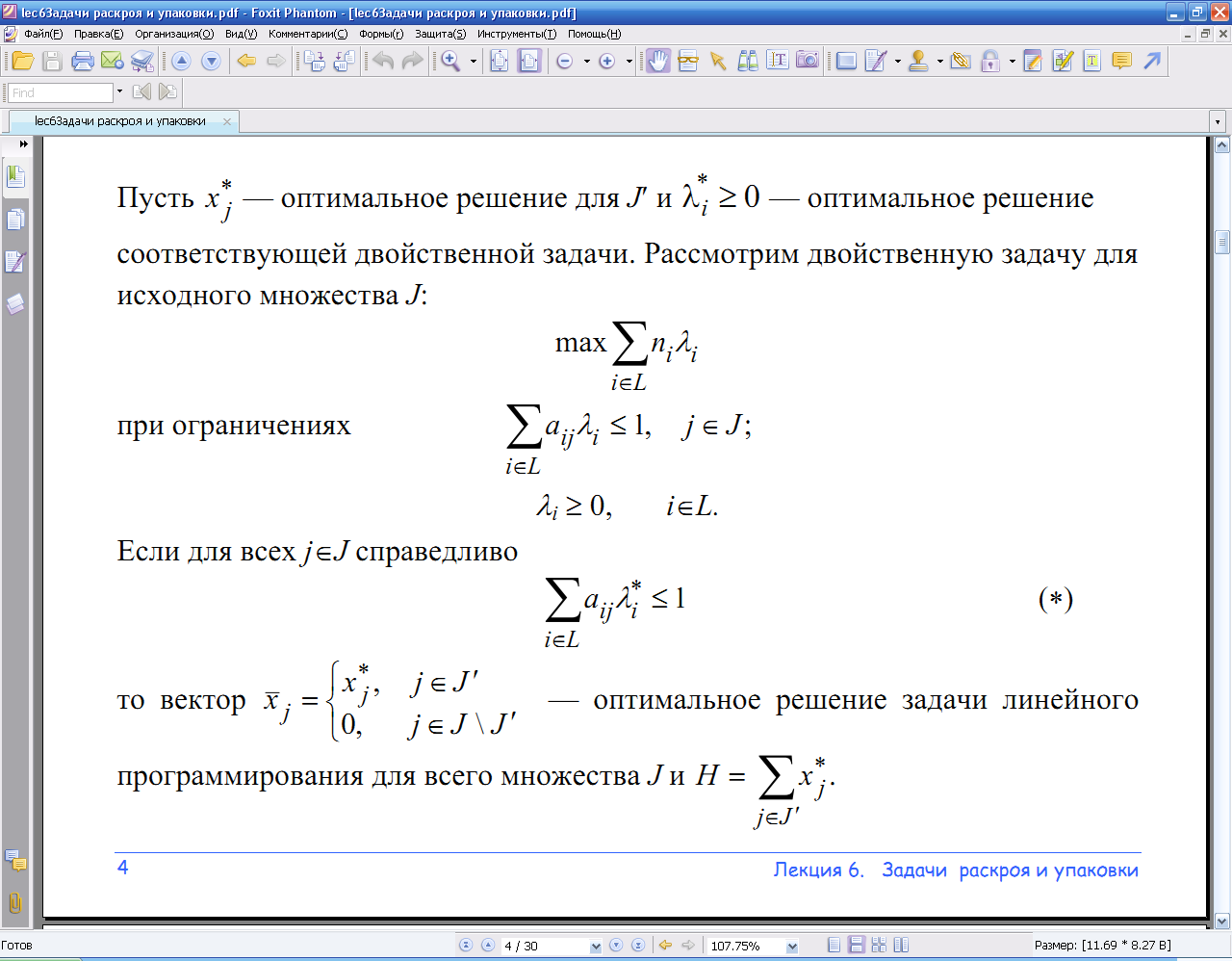
Это легко сделать с помощью любой жадной эвристики: NF, FF, BF,…

Пусть —  оптимальное решение для J′ и  — оптимальное решение соответствующей двойственной задачи. Рассмотрим двойственную задачу для

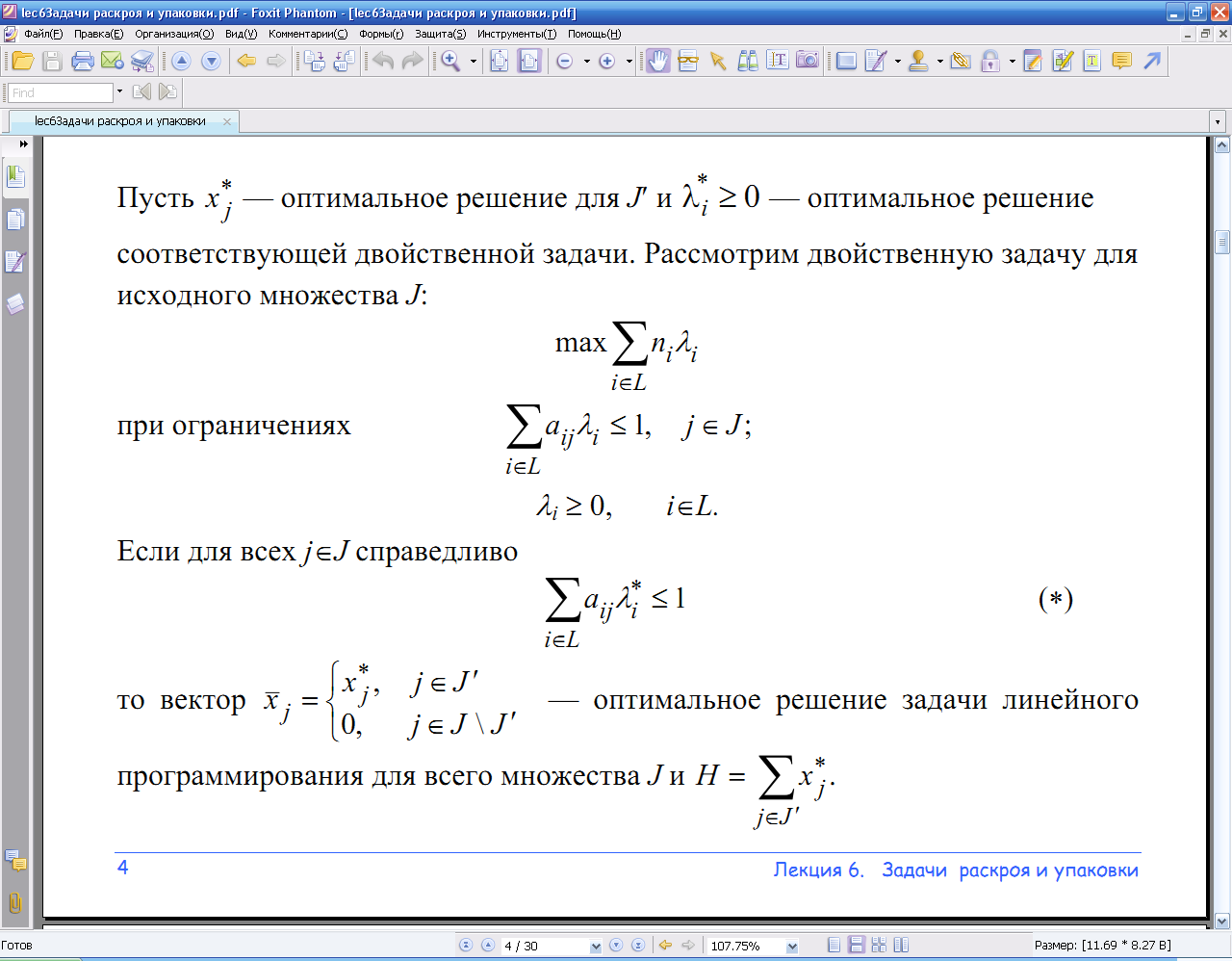
исходного множества J:



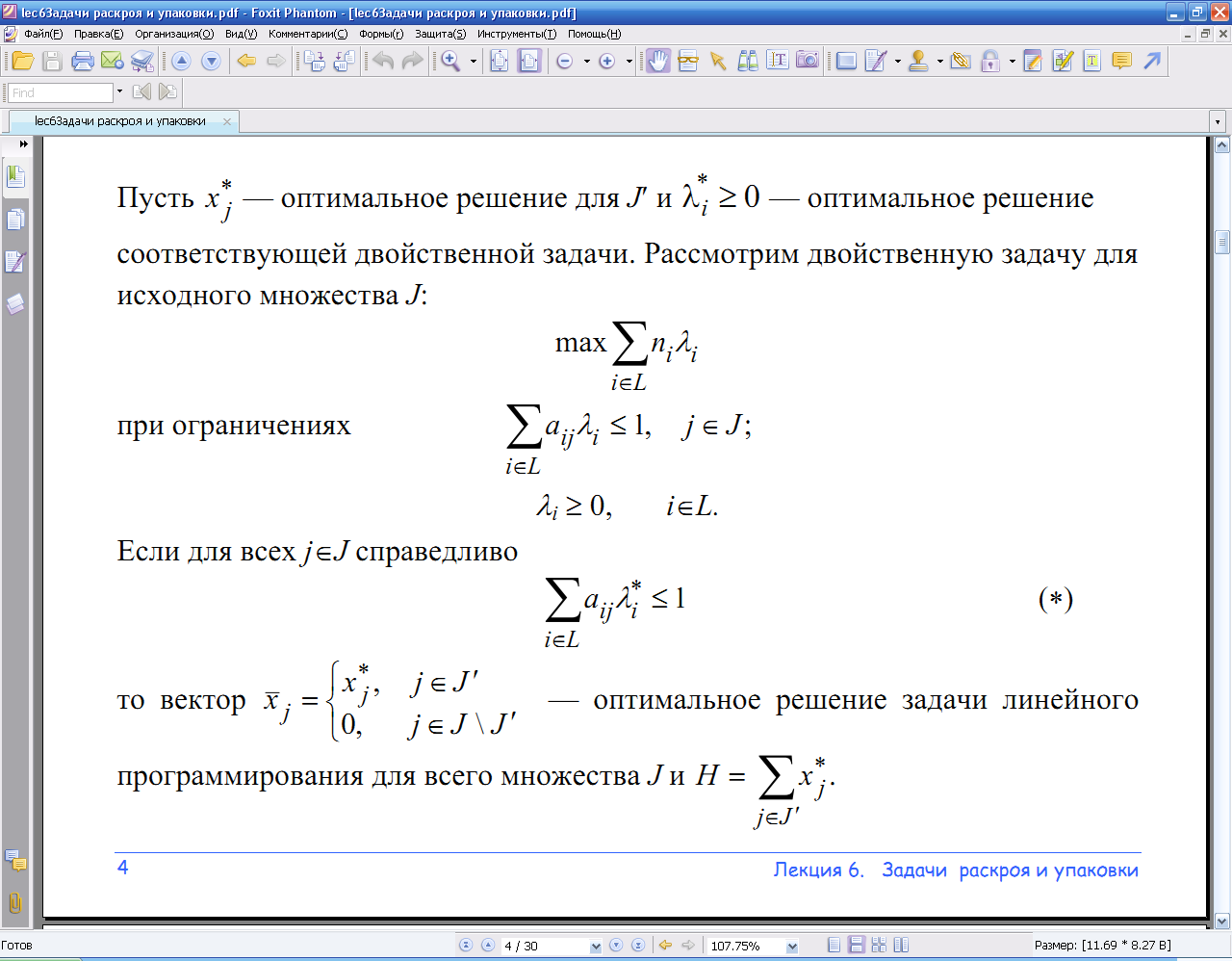
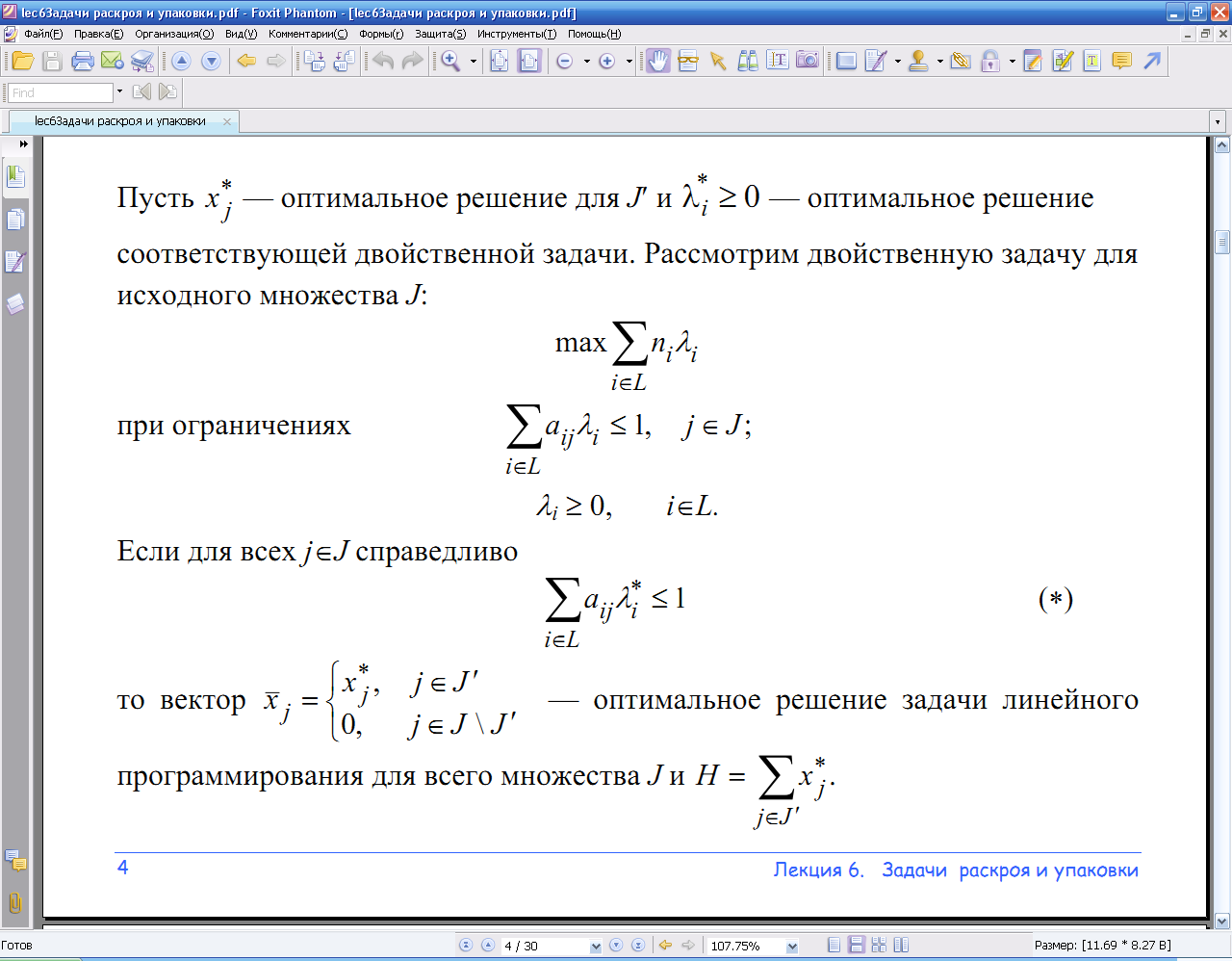
при ограничениях



Если для всех j∈J справедливо

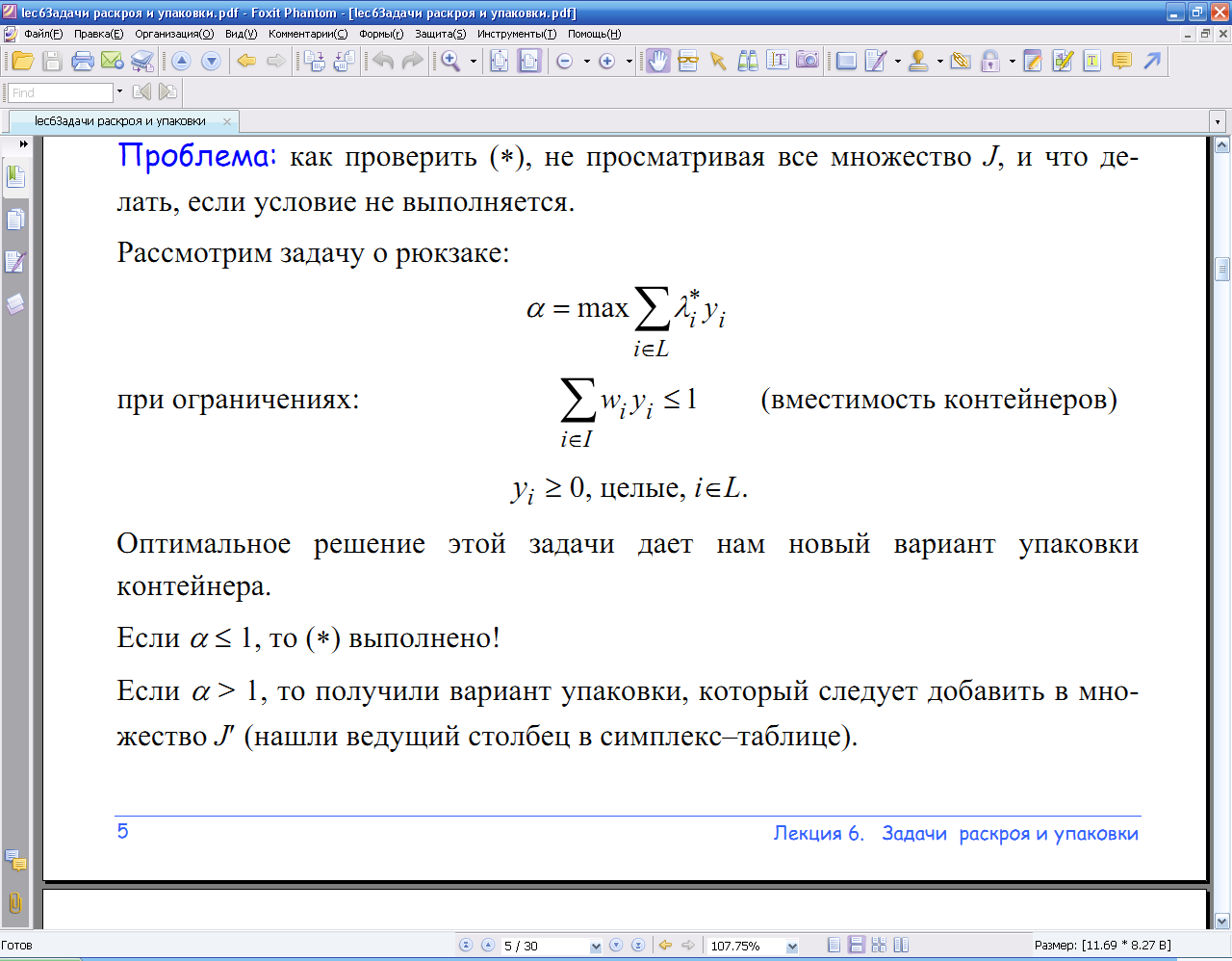
 (∗)

то вектор

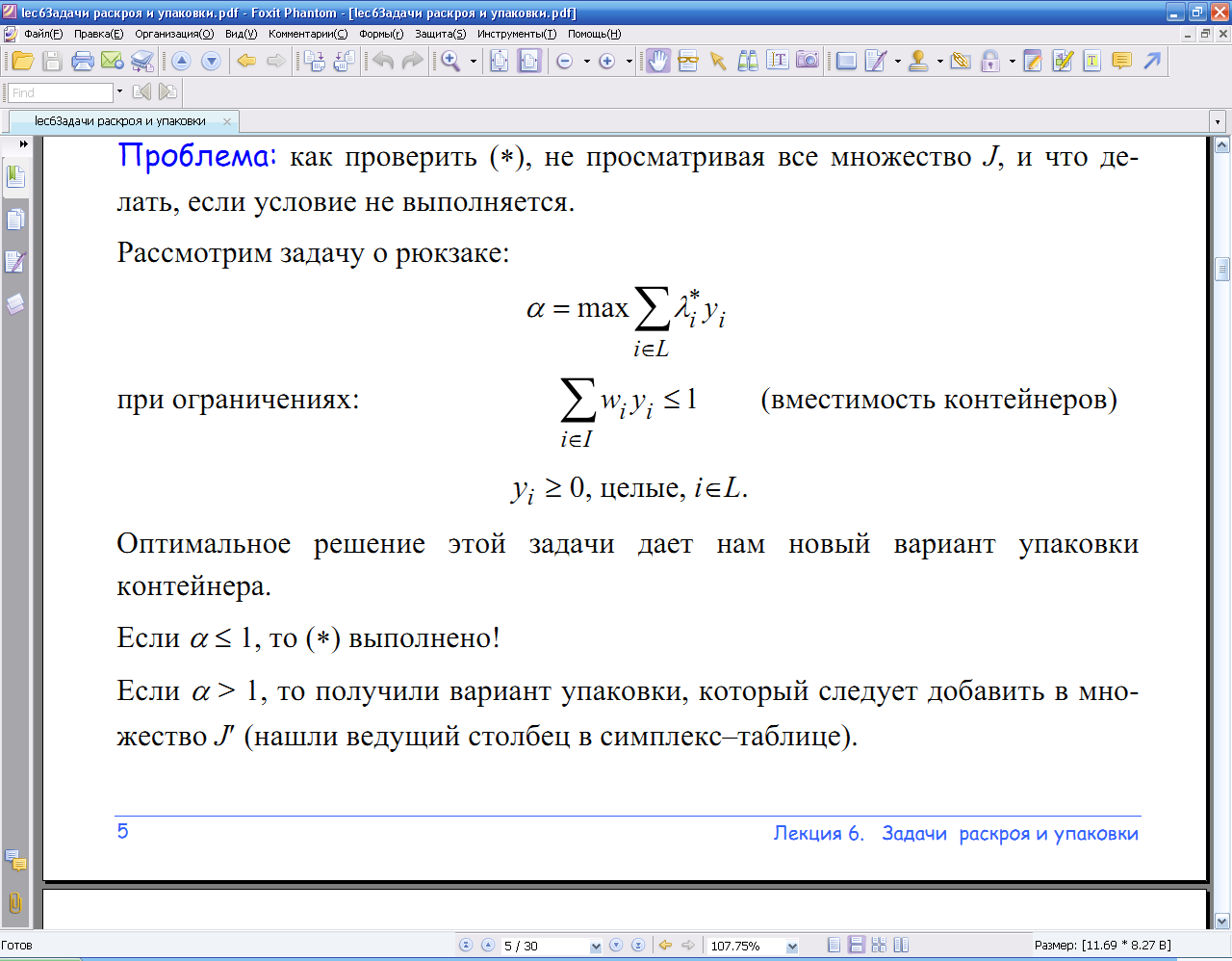
 — оптимальное решение задачи линейного программирования для всего множества J и .

Проблема: как проверить (∗), не просматривая все множество J, и что делать, если условие не выполняется.

Рассмотрим задачу о рюкзаке:



при ограничениях:

 (вместимость контейнеров)

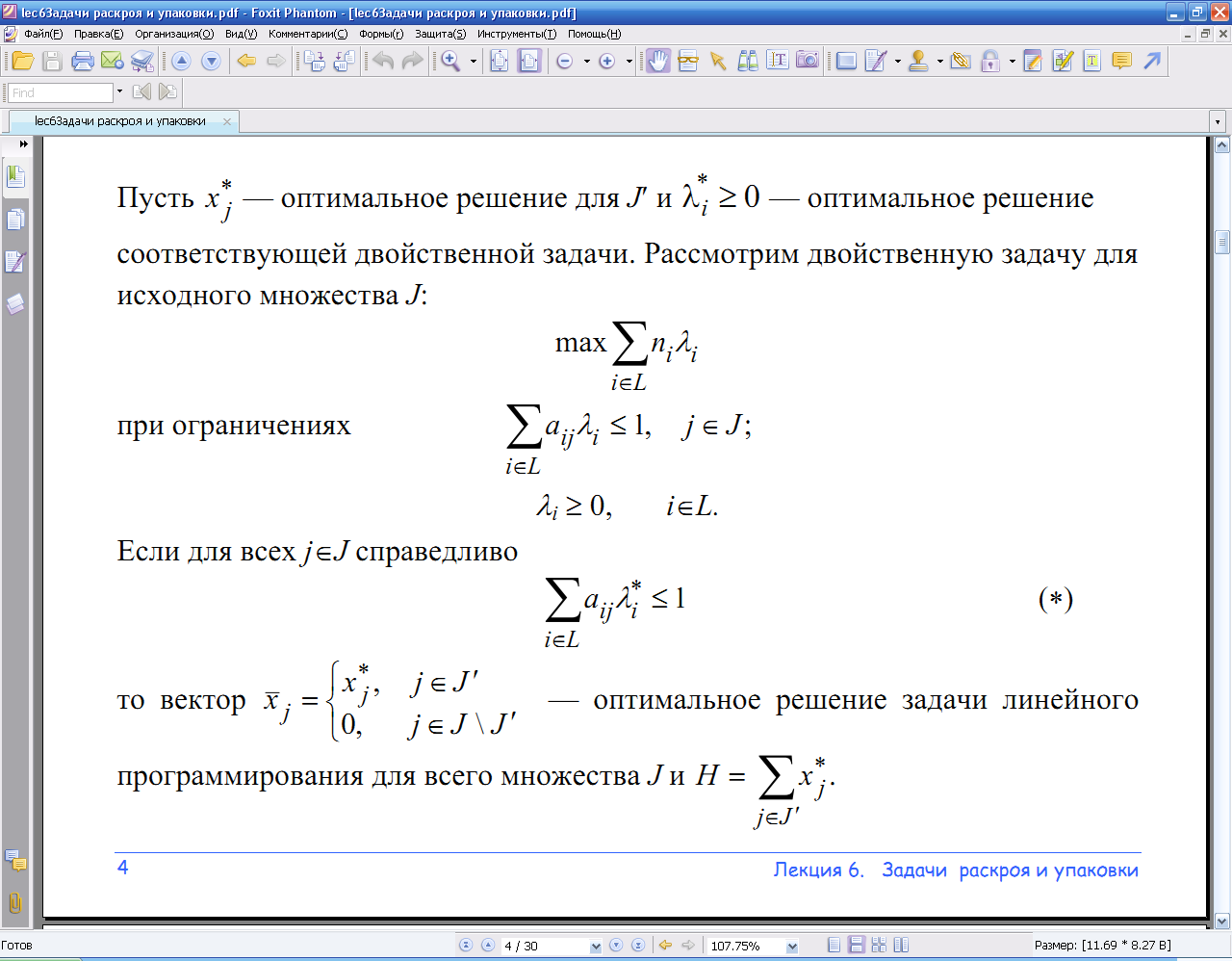
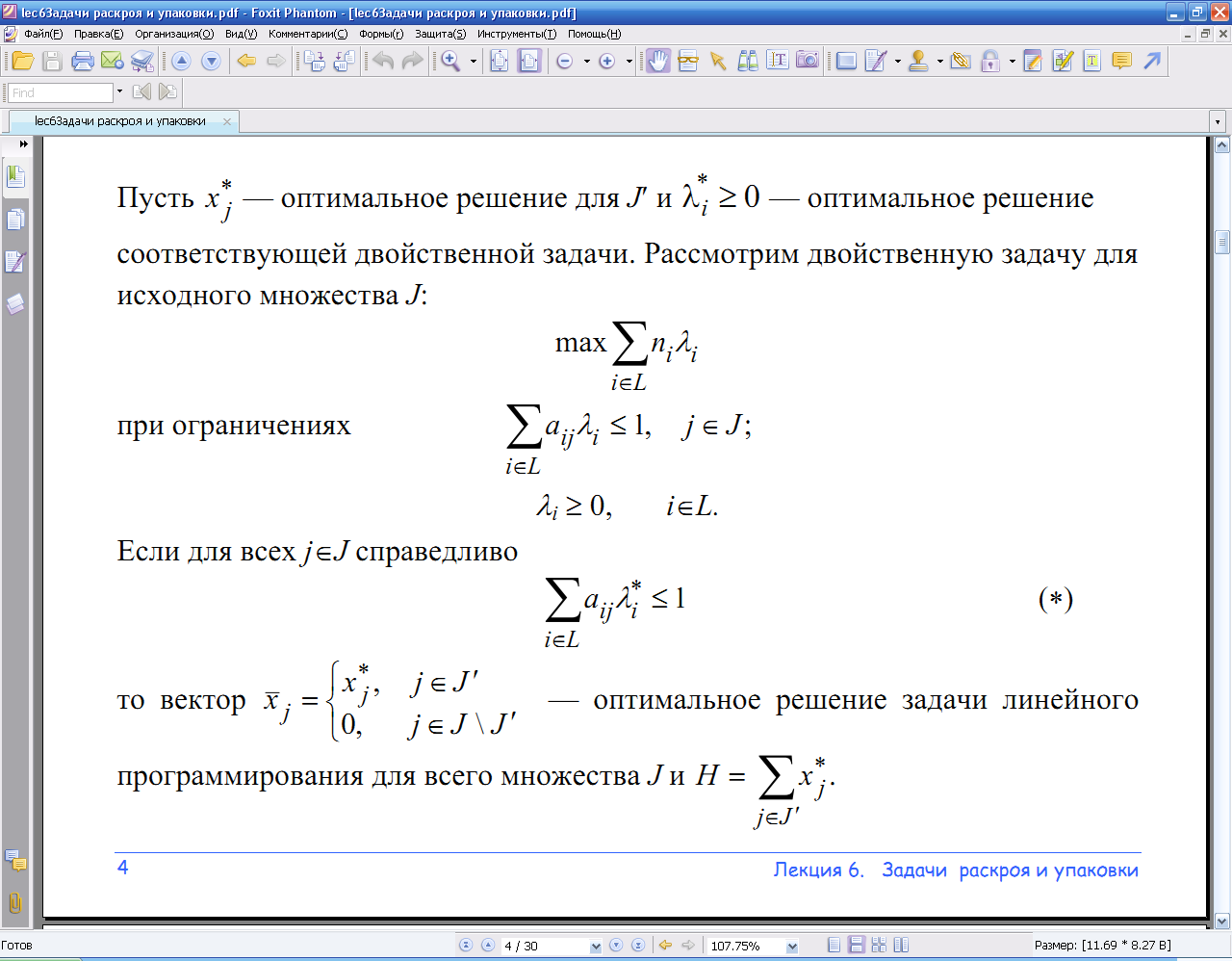
Оптимальное решение этой задачи дает нам новый вариант упаковки контейнера.

Если α ≤ 1, то (∗) выполнено!

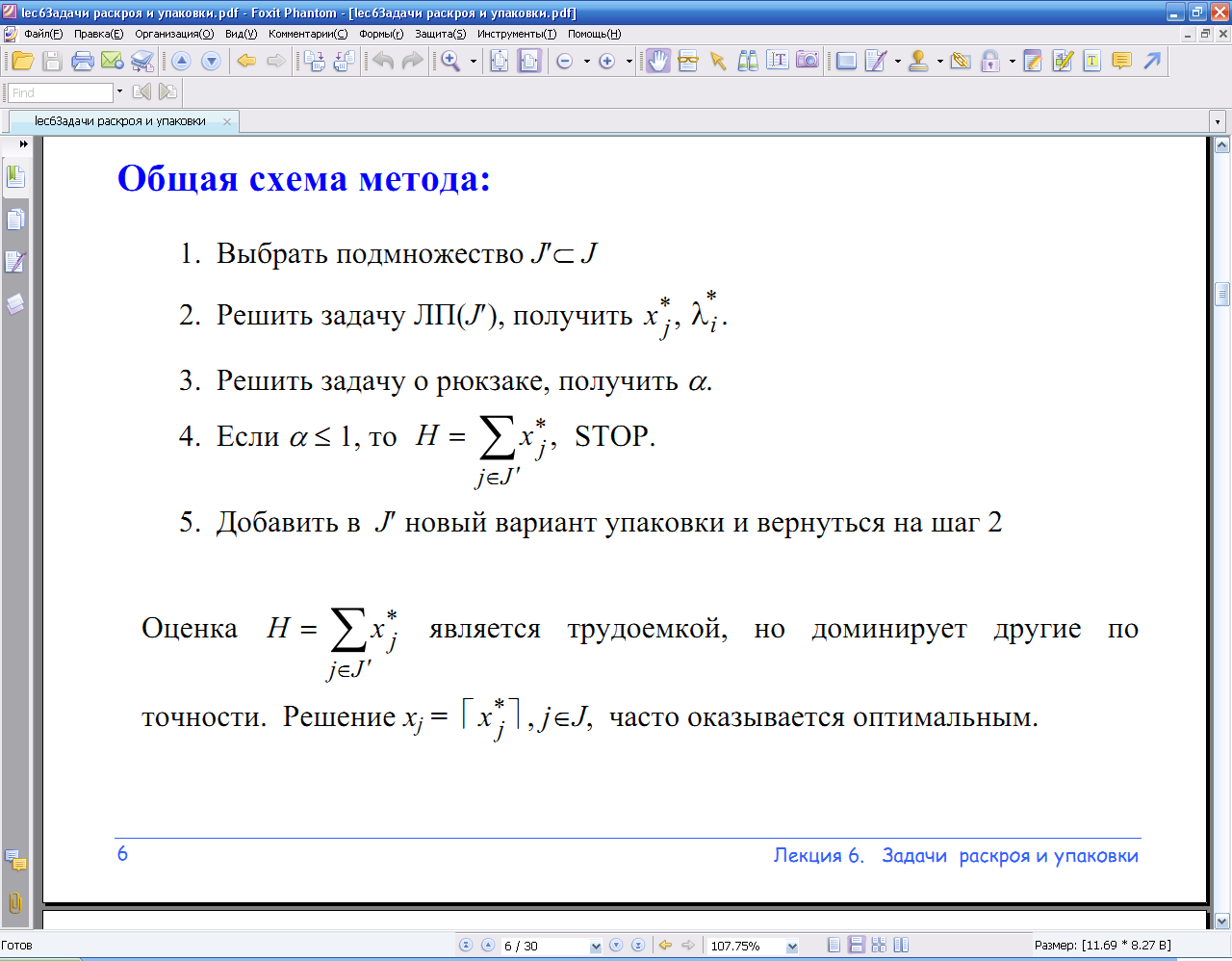
Если α > 1, то получили вариант упаковки, который следует добавить в множество J′ (нашли ведущий столбец в симплекс–таблице).

Общая схема метода:

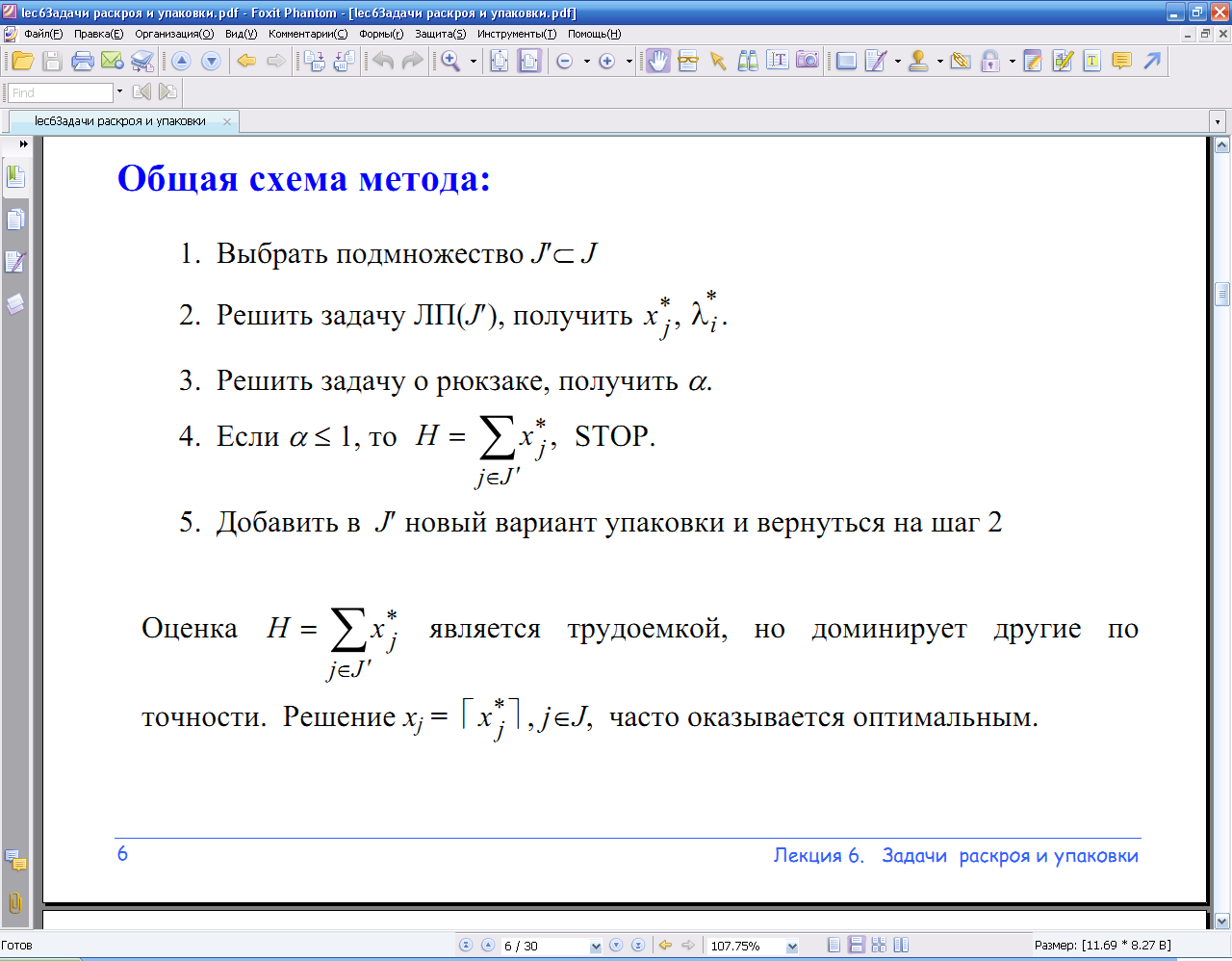
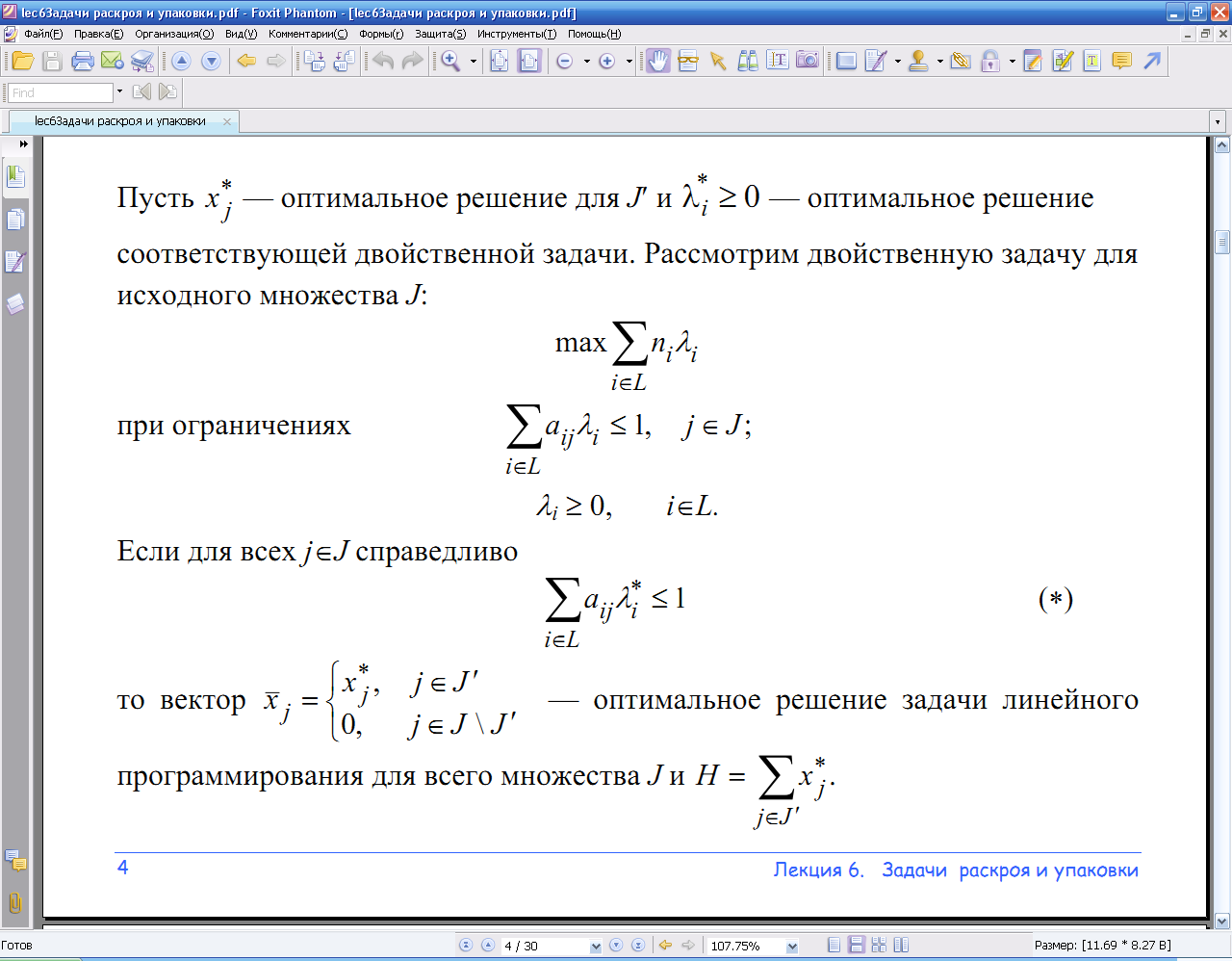
1. Выбрать подмножество J′⊂ J

2. Решить задачу ЛП(J′), получить ,.

3. Решить задачу о рюкзаке, получить α.

4. Если α ≤ 1, то 

5. Добавить в J′ новый вариант упаковки и вернуться на шаг 2

Оценка  является трудоемкой, но доминирует другие по точности. Решение xj = [] , j∈J, часто оказывается оптимальным.

2.6 Генетический алгоритм

Генети́ческий алгори́тм (англ. genetic algorithm) — это эвристический алгоритм поиска [10], используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. Является разновидностью эволюционных вычислений. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

|  |
| --- |
|  |

«Отцом-основателем» генетических алгоритмов считается Джон Холланд (en:John Henry Holland), книга которого «Адаптация в естественных и искусственных системах» (1975) является основополагающим трудом в этой области исследований. Ему же принадлежит доказательство теоремы схем.

Задача формализуется таким образом, чтобы её решение могло быть закодировано в виде вектора («генотипа») генов. Где каждый ген может быть битом, числом или неким другим объектом. В классических реализациях ГА предполагается, что генотип имеет фиксированную длину. Однако существуют вариации ГА, свободные от этого ограничения.

Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием «функции приспособленности», в результате чего с каждым генотипом ассоциируется определённое значение («приспособленность»), которое определяет насколько хорошо фенотип им описываемый решает поставленную задачу.

Из полученного множества решений («поколения») с учётом значения «приспособленности» выбираются решения (обычно лучшие особи имеют большую вероятность быть выбранными), к которым применяются «генетические операторы» (в большинстве случаев «скрещивание» — crossover и «мутация» — mutation), результатом чего является получение новых решений. Для них также вычисляется значение приспособленности, и затем производится отбор («селекция») лучших решений в следующее поколение.

Этот набор действий повторяется итеративно, так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (поколений), пока не будет выполнен критерий остановки алгоритма. Таким критерием может быть:

1. нахождение глобального, либо субоптимального решения;
2. исчерпание числа поколений, отпущенных на эволюцию;
3. исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в многомерных пространствах поиска.

Таким образом, можно выделить следующие этапы генетического алгоритма:

1. ) Задать целевую функцию (приспособленности) для особей популяции;
2. ) Создать начальную популяцию;
3. ) (Цикл);

Цикл имеет следующие пункты:

1. ) Размножение (скрещивание);
2. ) Мутирование;
3. ) Вычислить значение целевой функции для всех особей;
4. ) Формирование нового поколения (селекция);
5. ) Если выполняются условия останова, то (конец цикла), иначе (начало цикла).

Перед первым шагом нужно случайным образом создать начальную популяцию; даже если она окажется совершенно неконкурентоспособной, генетический алгоритм все равно достаточно быстро переведет ее в жизнеспособную популяцию. Таким образом, на первом шаге можно особенно не стараться сделать слишком уж приспособленных особей, достаточно, чтобы они соответствовали формату особей популяции, и на них можно было подсчитать функцию приспособленности (Fitness). Итогом первого шага является популяция H, состоящая из N особей.

Размножение в генетических алгоритмах обычно половое — чтобы произвести потомка, нужны несколько родителей, обычно два.

Размножение в разных алгоритмах определяется по-разному — оно, конечно, зависит от представления данных. Главное требование к размножению — чтобы потомок или потомки имели возможность унаследовать черты обоих родителей, «смешав» их каким-либо способом.

Почему особи для размножения обычно выбираются из всей популяции H, а не из выживших на первом шаге элементов H0 (хотя последний вариант тоже имеет право на существование)? Дело в том, что главный бич многих генетических алгоритмов — недостаток разнообразия (diversity) в особях. Достаточно быстро выделяется один-единственный генотип, который представляет собой локальный максимум, а затем все элементы популяции проигрывают ему отбор, и вся популяция «забивается» копиями этой особи. Есть разные способы борьбы с таким нежелательным эффектом; один из них — выбор для размножения не самых приспособленных, но вообще всех особей.

К мутациям относится все то же самое, что и к размножению: есть некоторая доля мутантов m, являющаяся параметром генетического алгоритма, и на шаге мутаций нужно выбрать mN особей, а затем изменить их в соответствии с заранее определенными операциями мутации.

На этапе отбора нужно из всей популяции выбрать определенную ее долю, которая останется «в живых» на этом этапе эволюции. Есть разные способы проводить отбор. Вероятность выживания особи h должна зависеть от значения функции приспособленности Fitness(h). Сама доля выживших s обычно является параметром генетического алгоритма, и ее просто задают заранее. По итогам отбора из N особей популяции H должны остаться sN особей, которые войдут в итоговую популяцию H'. Остальные особи погибают.

Генетические алгоритмы применяются для решения следующих задач:

1. Оптимизация функций;
2. Оптимизация запросов в базах данных;
3. Разнообразные задачи на графах (задача коммивояжера, раскраска, нахождение паросочетаний);
4. Настройка и обучение искусственной нейронной сети;
5. Задачи компоновки;
6. Составление расписаний;
7. Игровые стратегии;
8. Теория приближений;
9. Искусственная жизнь;
10. Биоинформатика (фолдинг белков);

2.6.1 Генетический алгоритм, применительно к задаче раскроя

Рассмотрим пример интерпритации генетического алгоритма для задачи раскроя.

Для генетического алгоритма основными понятиями являются хромосома, ген, популяция. Хромосома – одно из возможных решений, закодированное определенным образом. Хромосомы состоят из полей – генов. Хромосомы преобразуются за счет модификации генов. В ГА все операции производятся не над одной хромосомой, а над целым набором. Этот набор называется популяцией [20].

Направленный перебор решений осуществляется с помощью генетических операторов выбора родителей, кроссовера (сркрещивание), мутации, селекции, переупорядочивания.сначала формируется случайным образом исходное поколение, состоящее из k хромосом. Далее случайным образом или по какому-либо законусреди хромосом данного поколения выбираются пары родителей, причем вероятность выбора хромосом с лучшим значением ЦФ выше. Следующее поколение образуется из k перспективных дочерних хромосом, являющихся результатом ряда операций кроссовера. Кроссовер заключается в разрыва двух родительских хромосом и рекомбинирования оставшихся хромосомных отрезков. Мутации, т.е. те случайные изменения некоторых генов, происходят по некоторому правилу и служат для исключения зацикливания поиска в ограниченном подпространстве.

В качестве представления шаблона упаковки предлагается приоритетный список (Priority list, PL): p=p(1),p(2),…,p(m). P(i) – индексы прямоугольников. Перестановка – последовательность, в которой прямоугольники расположены на доске в соответсвии с определенными правилами. Также предполагается ,что p(i) имеет знак, который показывает направление размещения прямоугольника p(i) на доске.

Каждая родительская пара создает двух новых потомков с помощью оператора скрещивания:

Пусть pk, pj – два PL, соответствующихпаре родителей и pknew, pjnew – два новых PL, созданных скрещиванием. Сначалавыбирабтся два случайных числа p и q ,1<=p, q<=n. Начиная со случайной позиции p, скрещивающий оператор копирует q элементов из pk в pknew. Потом оставшиеся позиции в pknew заполняются другими элементами из pj в той же последовательности, в которой они встречаются в pj. Аналогично заполняется и pjnew.

Оператор мутации состоит из двух частей. Первая часть обменивает два случайных элемента или блока каждой новой перестановки с небольшой кероятностю pm1. Другая часть меняет знак каждого элемента в новых перестановках, т.е. поворачивает каждый прямоугольник на 900, с небольшой вероятностью pm2.

2.6.2 Генетический алгоритм с алгоритмами «нижний левый» и SVC

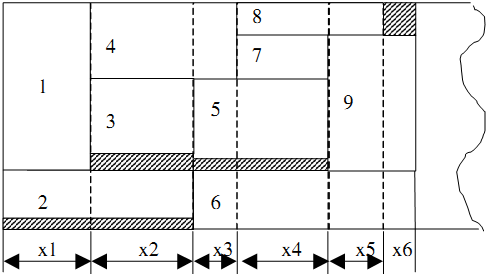
Генетический алгоритм берется в интерпритации раздела 2.6.1, изложенной выше.

Рассмотрим генетичсекий алгоритм ортогональной упаковки прямоугольников с применением алгоритма «нижний левый» как способа кодирования упаковок, и метод последовательного уточнения оценок, используемый в рамках генетического алгоритма [20].

Для оценки щаблона упаковки в ГА используется ЦФ, равная длине упакованной полосы.

При нахождении значения оценочной функции необходим алгоритм расшифровки перестановки P шаблона упаковки. Предлагается использовать IBL-алгоритм, который заключается в последовательном размещении прямоугольников, посредством их размещения, начиная с верхнего правого угла упакованной площади доски вниз, затем – влево и т.д. до тех пор, пока мы не сможем сделать ни одного перемещения, причем преимущество имеет движение вниз.

К полученным в ГА перестановкам применяется метод SVC. Для этого полоса, в которую упакованы прямоугольники по перестановке, полученной в ГА, разбивается параллельными основанию линиями на конечное число блоков. Каждый блок заканчивается с окончанием хотя бы одного из прямоугольников, входящих в него. Тогда блок рассматривается как совокупность стержней длины W, в которые укладываются одномерные предметы, количество стержней соответствует длине блока (рис. 2.6.1).



*Рисунок 2.6.1 – Разбиение упаковки на блоки*

Обозначим aij – количество i-х деталей в j-м блоке (раскрое), pj – остаток площади в j-v блоке.

Обозначим через xj – длину j-го блока, а через C – ширину остатка в этом блоке. Тогда площадь j-го блока Sj=W\*xjб, а площадь остатка в этом блоке pj= λj\*xj. Полезная площадь этого блока равна Sj-pj=W\*xj- λj\*xj и норма yij прямоугольника i в j-ом блоке с учетом остатка может быть вычислена по формуле

или

Учитывая, что прямоугольник I может быть расположен в нескольких блоках, получаем его норму

Где aij=1, если i-ый прямоугольник содержится в j-м блоке, и aij=0 в противном случае.

Строится PL по убыванию удельных оценок:

Далее полученный PL оценивается IBL-алгоритмом.

Затем выбираются k индивидов из всей совокупности родительского и полученного населения с наилучшими значениями ЦФ. Все эти k новых индивидов вместе составляют население потомков. Потом снова создаются новые потомки путем операторов скрещивания и мутации все эти процедуры повторяются до тех пор, пока заданная верхняя граница количества циклов не достигнет своего максимального значения или не будет получено ожидаемое значение ЦФ.

Результаты численных экспериментов показывают, что работа IBL&SVC дает лучшие результаты, по сравнению с алгоритмом GBA (Блочный Генетический Алгоритм) при большом значении относительных нижних и верхних весов длины прямоугольников.

2.7 Муравьиный алгоритм

Муравьиный алгоритм (алгоритм оптимизации подражанием муравьиной колонии, англ. ant colony optimization, ACO) — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжера, а также аналогичных задач поиска маршрутов на графах [10]. Суть подхода заключается в анализе и использовании модели поведения муравьев, ищущих пути от колонии к источнику питания и представляет собой метаэвристическую (англ. methaheuristic, meta — «за пределами» и heuristic — «найти») оптимизацию. Изначально предложен доктором философских наук Марко Дориго в 1992 году, является первым алгоритмом, направленым на поиск оптимального пути в графе.

В основе алгоритма лежит поведение муравьиной колонии— маркировка более удачных путей большим количеством феромона. Работа начинается с размещения муравьёв в вершинах графа (городах), затем начинается движение муравьёв — направление определяется вероятностным методом, на основании формулы вида: , где: Pi вероятность перехода по пути i, li величина, обратная весу (длине) i-ого перехода, fi количество феромона на i-ом переходе, q величина, определяющая «жадность» алгоритма, p величина, определяющая «стадность» алгоритма и q + p = 1

Решение не является точным и даже может быть одним из худших, однако, в силу вероятностности решения, повторение алгоритма может выдавать (достаточно) точный результат.

В литературе было предложено несколько метаэвристических моделей ACO. Среди них три наиболее успешные:

1) ant system (Dorigo 1992, Dorigo et al. 1991, 1996);

2) ant colony system (ACS) (Dorigo & Gambardella 1997);

3) MAX-MIN ant system (MMAS) (Stutzle & Hoos 2000);

В реальном мире, муравьи (первоначально) ходят в случайном порядке и по нахождению продовольствия возвращаются в свою колонию, прокладывая феромонами тропы. Если другие муравьи находят такие тропы, они, вероятнее всего, пойдут по ним. Вместо того, чтобы отслеживать цепочку, они укрепляют её при возвращении, если в конечном итоге находят источник питания. Со временем феромонная тропа начинает испаряться, тем самым уменьшая свою привлекательную силу. Чем больше времени требуется для прохождения пути до цели и обратно, тем сильнее испарится феромонная тропа. На коротком пути, для сравнения, прохождение будет более быстрым и как следствие, плотность феромонов остаётся высокой. Испарение феромонов так же имеет свойство избежания, стремления к локально-оптимальному решению. Если бы феромоны не испарялись, то путь, выбранный первым, был бы самым привлекательным. В этом случае, исследования пространственных решений были бы ограниченными. Таким образом, когда один муравей находит (например, короткий) путь от колонии до источника пищи, другие муравьи, скорее всего пойдут по этому пути, и положительные отзывы в конечном итоге приводят всех муравьёв к одному, кратчайшему, пути.

Оригинальная идея исходит от наблюдения за муравьями в процессе поиска кратчайшего пути от колонии до источника питания.

1. Первый муравей находит источник пищи (F) любым способом (а), а затем возвращается к гнезду (N), оставив за собой тропу из феромонов (b);
2. Затем муравьи выбирают один из четырёх возможных путей, затем укрепляют его и делают привлекательным;
3. Муравьи выбирают кратчайший маршрут, так как у более длинных феромоны сильнее испарились.

Среди экспериментов по выбору между двумя путями неравной длины, ведущих от колонии к источнику питания, биологи заметили, что, как правило, муравьи используют кратчайший маршрут. Модель такого поведения заключается в следующем:

1. Муравей (так называемый «Блиц») проходит случайным образом от колонии;
2. Если он находит источник пищи, то возвращается в гнездо, оставляя за собой след из феромона;
3. Эти феромоны привлекают других муравьёв находящихся вблизи, которые вероятнее всего пойдут по этому маршруту;
4. Вернувшись в гнездо они укрепят феромонную тропу;
5. Если существует 2 маршрута, то по более короткому, за то же время, успеют пройти больше муравьёв, чем по длинному;
6. Короткий маршрут станет более привлекательным;
7. Длинные пути, в конечном итоге, исчезнут из-за испарения феромонов.

Муравьи используют окружающую среду как средство общения. Они обмениваются информацией косвенным путём, через феромоны, в ходе их «работы». Обмен информации имеет локальный характер, только те муравьи, которые находятся в непосредственной близости, где остались феромоны — могут узнать о них. Такая система называется «Stigmergy» и справедлива для многих социальных животных (был изучен в случае строительства столбов в гнёздах термитов). Данный механизм решения проблемы очень сложен и является хорошим примером самоорганизации системы. Такая система базируется на положительной (другие муравьи укрепляют феромонную тропу) и отрицательной (испарение феромонной тропы) обратной связи. Теоретически, если количество феромонов будет оставаться неизменным, с течением времени, по всем маршрутам, невозможно будет выбрать путь. Однако из-за обратной связи, небольшие колебания приведут к усилению одного из маршрутов и система стабилизируется к кратчайшему пути.

3 Обзор аналогичных программных систем и выбор инструментального средства разработки

3.1 Описание аналогичных программных систем

Существуют программы, занимающиеся поиском оптимальных решений для задач линейного программирования. Это такие программы как: PROSA, TORA, МИО 2004, и другие. Рассмотрим только некоторые из них.

3.1.1 Система «Техтран-Раскрой»

Программа предназначена для комплексного решения задач раскроя листовых материалов. Она сочетает возможности системы подготовки управляющих программ с функциями организации производственного процесса. Возможности данной систамы позволяют:

1. построение геометрической модели детали;
2. обмен геометрическими данными с другими CAD-системами;
3. использование базы данных и делового отхода (формируется из неиспользованной части листа и добавляется базу данных для последующего использования);
4. формирование задания на раскрой;
5. автоматическое и ручное размещение деталей на листе;
6. разделка кромок под сварку;
7. документирование карт и спецификаций раскроя;
8. получение управляющих программ для различных станков с ЧПУ;
9. настройка программы на конкретное оборудование с ЧПУ.

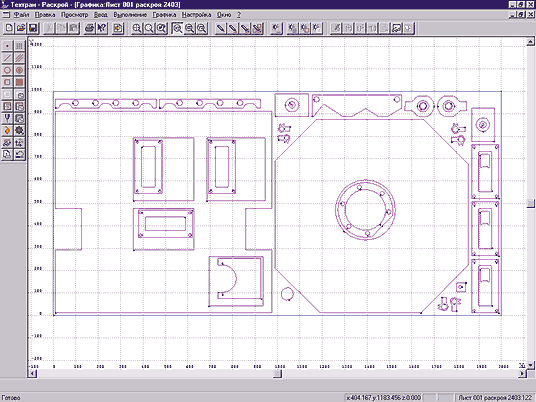


Рисунок 3.1-Система «Техтран-Раскрой»

3.1.2 Программа «Астра Раскрой»

Программа раскроя Астра Раскрой разработана специально для мебельных предприятий. Может использоваться также для раскроя стекла и металла. Программа Астра Раскрой включает специальные функции для сервисных центров по раскрою листовых материалов.

Программа раскроя Астра Раскрой предназначена для оптимизации раскроя листовых материалов - древесностружечных плит, металла, стекла и пластиков. Возможности программы позволяют:

1. формирование заказа для раскроя;
2. автоматический раскрой заказа;
3. редактирование карты раскроя;
4. рассчет, сохранение и использование мерных остатков после раскроя;
5. получнение и распечатка технической документации в полном объеме;
6. рассчет стоимости заказа и распечатка счета-фактуры;

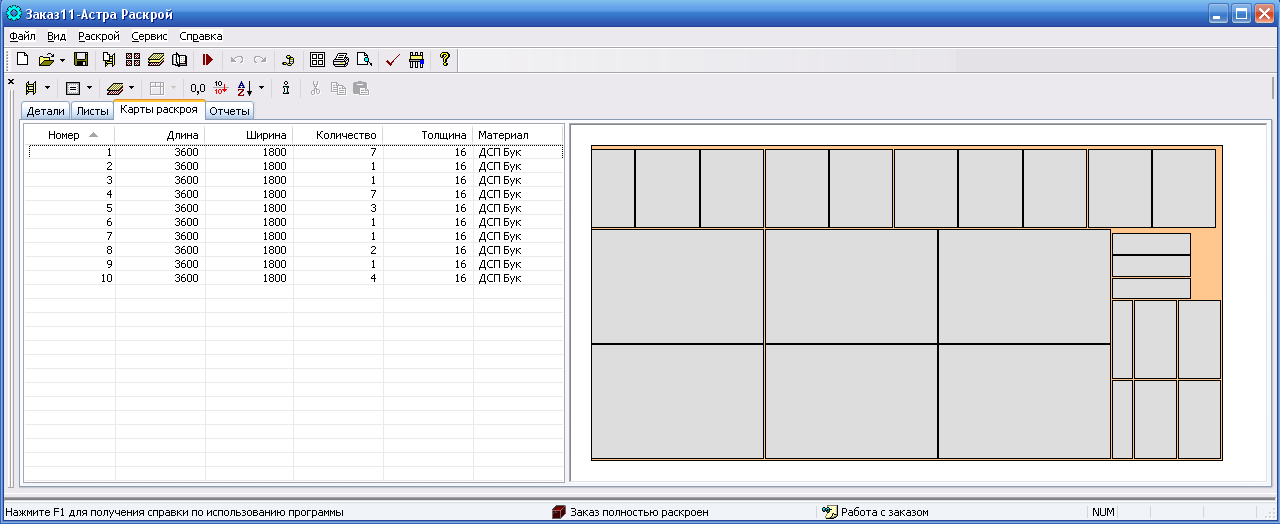


Рисунок 3.-Интерфейс программы «Астра Раскрой»

3.1.3 Сапр «Интех-Раскрой»

Программа «Интех-Раскрой» предназначена для автоматизация создания технологии раскроя и управляющих программ для газокислородных, плазменных, лазерных станков, гильотинных ножниц.

Возможности данной систамы позволяют:

1. построение геометрической модели детали;
2. обмен геометрическими данными с другими CAD-системами;
3. использование базы данных и делового отхода;
4. формирование задания на раскрой;
5. автоматическое и ручное размещение деталей на листе;
6. получение управляющих программ для различных станков с ЧПУ;
7. многопользовательскую работу и «доступ» к сетевым ресурсам;
8. создание многооперационного технологического процесса.
9. многоинструментальную обработку в параллельном и последовательном режиме;
10. расчет оптимальных маршрутов холостых ходов;
11. включение в УП маркировки, разметки, сверления.
12. генерацию подробных отчетов, спецификаций;
13. калькуляцию себестоимости обработки;
14. моделирование процесса обработки деталей в модуле Верификатор;
15. расчет траекторий движения наклонного резака в 3D
16. выполнение разгибок 3D поверхностей;
17. построение разверток сечений 3D элементов трубопроводов;
18. анализ сводной информации по проектам раскроя
19. проектировать раскрой на гильотинных ножницах;

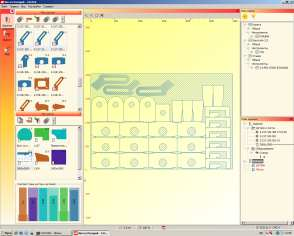


Рисунок 3.-Интерфейс Сапр «Интех-Раскрой»

3.1.4 Программа «2D-Place»

Программа 2D-Place осуществляет раскрой листового материала.

**Программа позволяет:**

1. производить автоматический расклад деталей по листам (с учетом направления фактуры), с целью минимизации отходов, и, при необходимости, с учетом содержимого склада стандартных листов и полезных отходов;
2. вести склад стандартных листов и остатков с учётом их материалов;
3. вести базы деталей с учетом направления волокон, материала и кромки;
4. вести базы материалов и кромок;
5. распечатывать карты раскроя и статистику разложения;
6. хранить базу моделей мебели;
7. работать с базами деталей предыдущих версий программы и файлами универсальных форматов (с разделителями табуляциями).
8. задавать параметры полезных отходов для каждого из материалов;
9. распечатывать базы деталей на принтере и передавать в редактор электронных таблиц MS Excel;

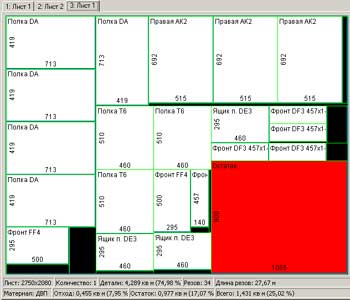


Рисунок 3.-Карта раскроя,полученная в программе 2D-Place

3.1.5 Сапр «Сириус»

СИРИУС (Система Интерактивного Раскроя И Управления Станками) - это новая высокоэффективная разработка, предназначенная для проектирования раскроя материала на заготовки произвольной конфигурации и подготовки управляющих программ для машин термической резки металла. Кроме того, СИРИУС - это универсальное средство для проектирования управляющих программ для обработки деталей на токарных и фрезерных станках с ЧПУ (класса 2.5 D), в том числе, на всех типах сверлильных.

СИРИУС реализован на IBM - совместимых персональных компьютерах в среде MS-DOS и WINDOWS. C 1996 г. поставляется новая версия САПР СИРИУС, которая обладает целым рядом принципиально новых возможностей, делающих ее сравнимой, а по целому ряду характеристик значительно превосходящей самые лучшие зарубежные разработки такого рода.

Отличительные особенности СИРИУСа:

1. эффективное кодирование деталей любой сложности;
2. не имеющий аналогов по своим функциональным возможностям мощный
3. специализированный графический редактор, обеспечивающий проектирование раскройной карты с максимально возможным коэффициентом использования материала за минимальное время;
4. целый спектр новейших алгоритмов автоматического и автоматизированного раскроя;
5. учет всех технологических особенностей резки и возможностей оборудования для резки;
6. возможность использования подпрограмм;
7. органичная стыковка с устройствами ввода-вывода на перфоленту. Вывод эскизов раскройной карты на все типы принтеров и плоттеров;
8. встроенный интерпретатор языка С;
9. уникальный сервис для пользователя.

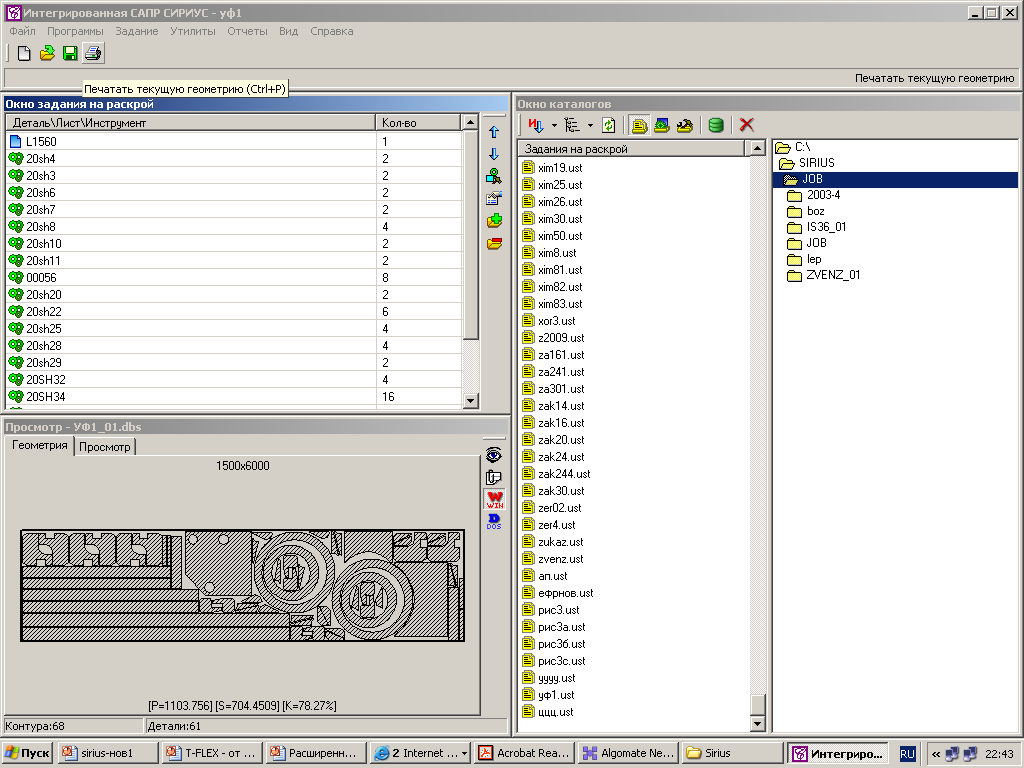


Рисунок 3.-Интерфейс генератора заданий Сапр «Сириус»

3.1.6 Программа «T-Flex/Раскрой»

Программа **T-FLEX/Раскрой** предназначена для расчета и построения эскизов оптимальных схем раскроя листового материала и ориентирована на решение следующих задач:

1. раскрой листов на карты и/или полосы;
2. раскрой произвольной плоской детали в полосе и/или листе, так называемый регулярный раскрой;
3. раскрой группы разнородных деталей в произвольно заданной форме плоской заготовки, так называемый нерегулярный или фигурный раскрой.

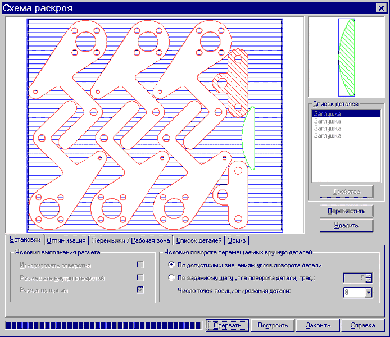


Рисунок 3.6-Интерфейс программы «T-Flex/Раскрой»

3.1.7 САПР «AccuMark»

Сапр для конструирования, градации, раскладки и подготовки кроя.

Основные особенности;

1. Ввод лекал, используя дигитайзер или формат PDS;
2. Гибкие возможности градации;
3. Создание лекал при помощи основных функций конструктора;
4. Управление данными включая резервное копирование, восстановление данных, и хранение данных;
5. Генерация отчетов;
6. Быстрый и эффективный ввод данных;
7. Горячие клавиши и расширенный пользовательский интерфейс;
8. Изготовление раскладки, включая возможность КОПИРОВАНИЯ раскладки;
9. Интеллектуальная раскладка лекал;
10. Вывод лекал и раскладок на плоттер и управление очередью плоттера;
11. Доступные языки: Китайский, Английский, Португальский, Испанский, Русский.

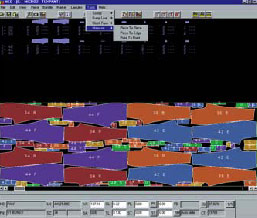


Рисунок 3.7-Интерфейс САПР «AccuMark»

3.1.8 Программа «OptiTex»

OptiTex - Программа для проектирования одежды и управления раскроем тканей, разработанные специально для среды Windows. Достойной упоминания является возможность Optitex Runway одевать на виртуальный манекен одновременно несколько изделий. При этом будут учтены различные нюансы механического и визуального взаимодействия соприкасающихся частей изделий.

В программу OptiTex входят:

1. Дигитализация. Ввод лекал в виде контурных изображений, в компьютер.
2. Конструирование шаблонов. Модельер разрабатывает детали джинсов непосредственно в компьютере.
3. Градация. Программа размножает шаблон по размерам и ростам.
4. Runway Designer - 3D- модуль, максимально приближенный к реальности, обеспечивает возможность моделирования одежды на манекенах различных форм и размеров. Модуль Optitex Runway может одевать на виртуальный манекен одновременно несколько изделий. При этом будут учтены взаимодействия соприкасающихся частей изделий. В следующей версии система будет в динамике одевать изделие на движущуюся фигуру манекена.
5. Раскладка лекал. Программа размещает лекала так, чтобы общая длина раскладки максимально уменьшилась.

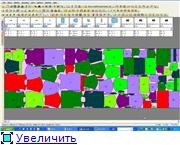


Рисунок 3.8-Интерфейс программы «OptiTex»

4 Разрабонтанное приложение

Основное рабочее окно программы представлено на рисунке 4.1.

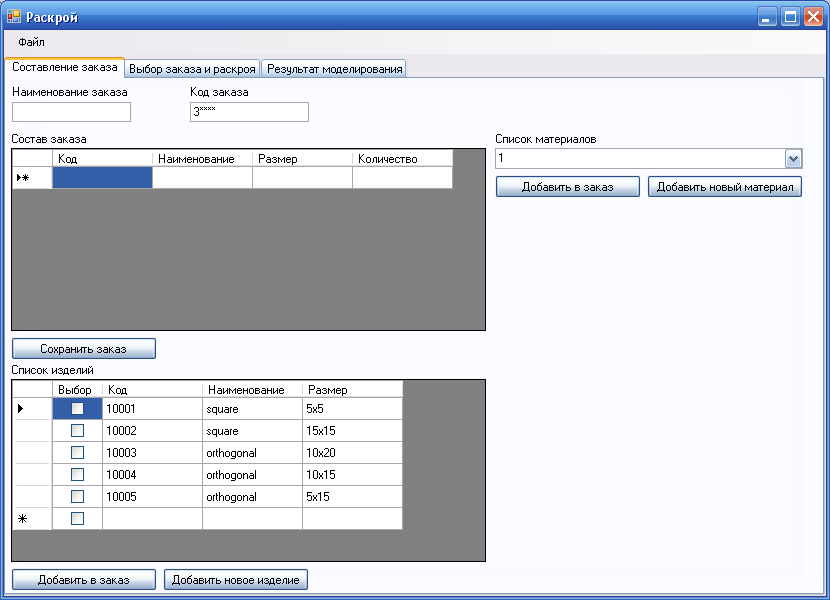


Рисунок 4.1-Интерфейс приложения «Раскрой»

На рисунке 4.2 изображены схемы таблиц баз данных, используемые в приложении «Расрой».

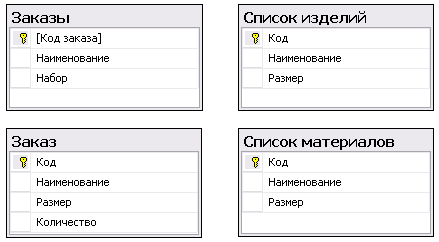


Рисунок 4.2-Таблицы баз данных приложения

На рисунке 4.3 показана вкладка «Составление заказа», на которой отображаюся «Список материалов» и «Список изделий». В данных списках пользователь выбирает нужные элементы и добавляет их, после ввода наименования и код заказа, по средствам кнопки «Добавить в заказ». С помощью соответствующих кнопок списки можно добавить новые изделия и материалы.

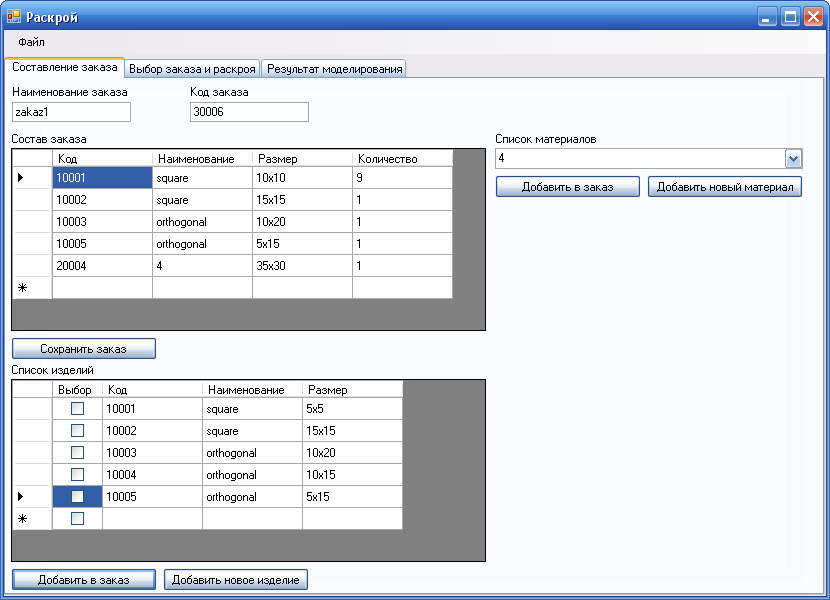


Рисунок 4.3-Вкладка «Составление заказа»

На рисунке 4.4 представлена вкладка «Выбор заказа», на которой осуществляется выбор заказа для раскроя, а так же метод раскроя. При нажатии на кнопку «Рассчитать раскрой» осуществляется моделирование раскроя.

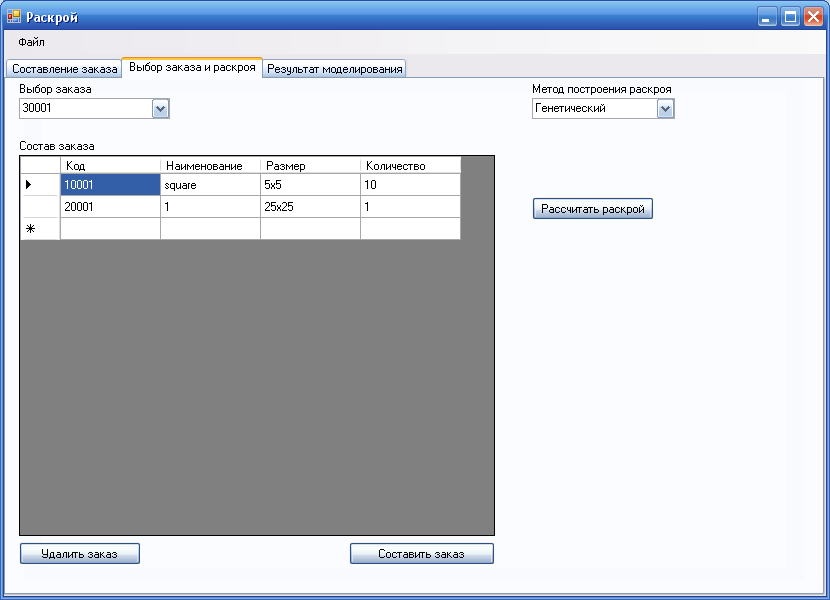


Рисунок 4.4-Вкладка «Выбор Заказа»

Вкладка с отображением результатов моделирования представлена на рисунке 4.5.

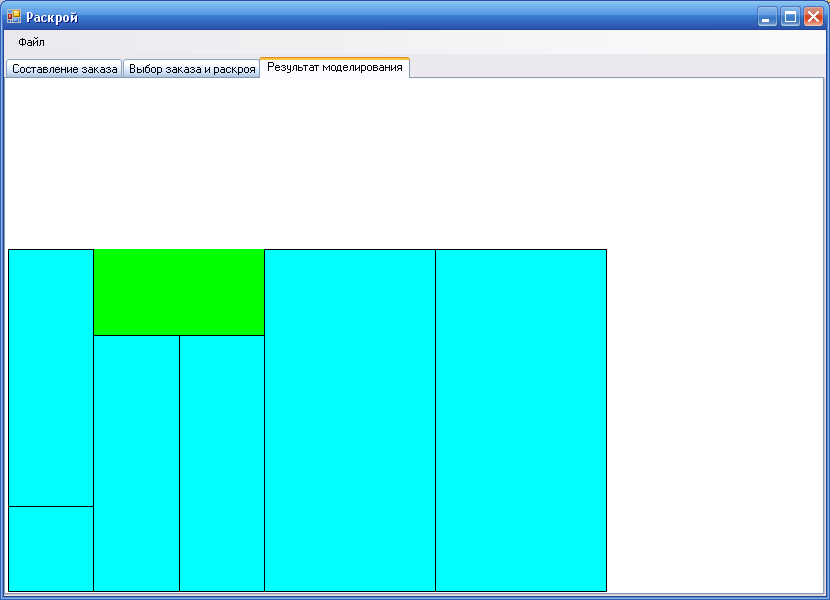


Рисунок 4.5-Вкладка «Результат моделирования»

На рисунках 4.6 и 4.7 представлены результаты проектировнаия раскроя одного набора деталей с помощью алгоритмов «Левый нижний» и «Генетический алгоритм» с декодером «Левый нижний», соответственно.

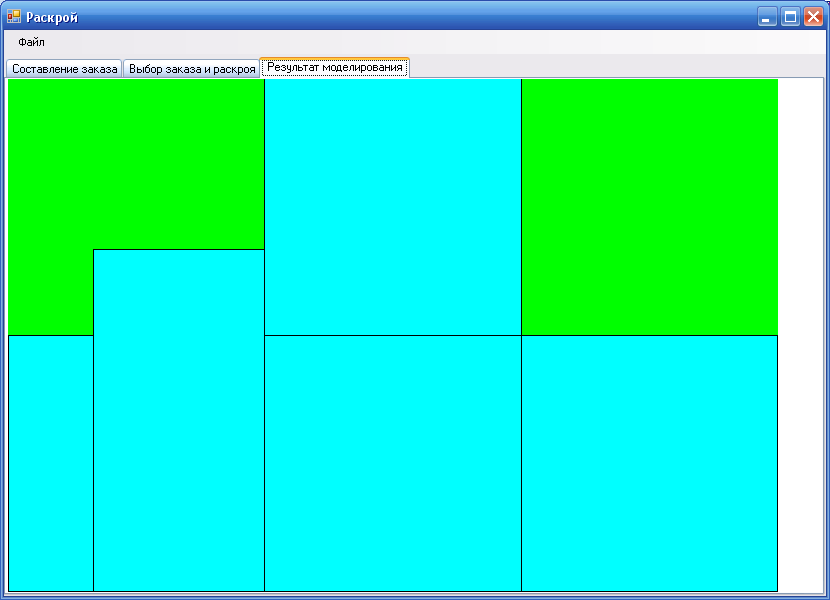


Рисунок 4.6-Алгоритм «Левый нижний»

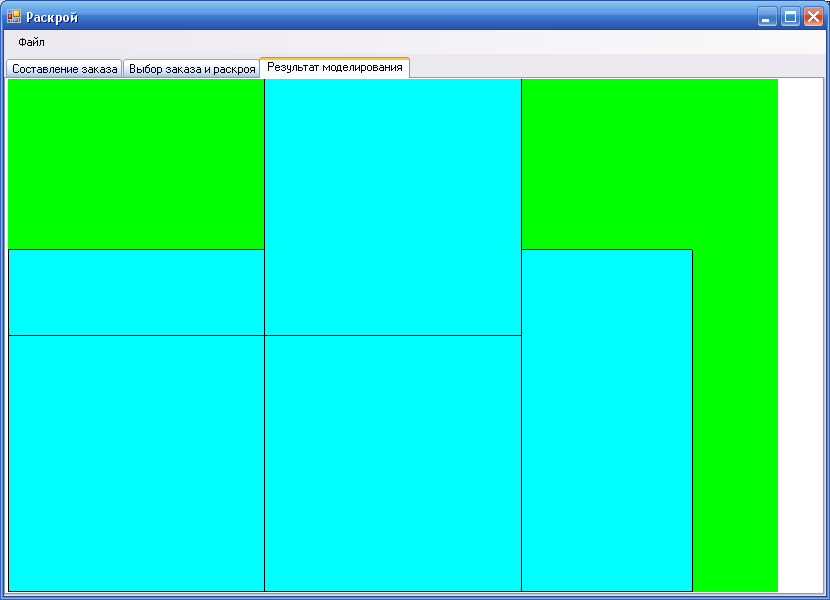


Рисунок 4.7-«Генетический алгоритм» с декодером «Левый нижний»

Список использованных источников

1. Фроловский В. Д. Автоматизация проектирования управляющих программ. Учебное пособие / Фроловский В. Д. – Новосибирск: ООО Издательский центр «Прогресс-сервис», 2006. – 98 с.
2. Фроловский В. Д. Приближенные методы решения NP-трудных задач в системах автоматизации проектирования : учеб. пособие / Фроловский В. Д. - Новосибирск, НГТУ, 2006. - 100 с.
3. Фроловский В. Д. Избранные задачи геометрического проектирования. Пара-метризация сложных поверхностей : учеб. пособие / Фроловский В. Д. - Новосибирск: Издательский центр «Прогресс-сервис», 2005. - 165 с.
4. Фроловский В. Д. Автоматизация проектирования систем рационального использования материалов (задачи, методы, приложения) / В. Д. Фроловский // Автоматизированные системы и информационные технологии : сб. науч. тр. Рос. науч.-практ. конф. (Новосибирск, 22-23 сент. 2011 г.). – Новосибирск : Изд-во НГТУ, 2011. – C. 231–253.
5. Фроловский В. Д. Оптимальное группирование геометрических объектов при проектировании карт раскроя материалов / В. Д. Фроловский // Материалы 9-й и 10-й Всероссийской научно- практ. конф. по графическим информационным технологиям и системам. Ч. 1. Н.-Новгород. НГТУ. 2000. С. 60-64
6. Мухачева Э. А., Мухачева А. С. Задача прямоугольной упаковки: методы локального поиска оптимума на базе блочных структур: Автоматика и телемеханика, №2. – 2004. – 112 с.
7. Руднев А. С. Вероятностный поиск с запретами для задачи упаковки кругов и прямоугольников в полосу: Дискретный анализ и исследование операций, июль-август 2009 Том 16, №4. – 2009. – 61-86 с.
8. Петренко С. В. Оптимизация размещения двумерных геометрических объектов на анизотропном материале с использованием методов математического программирования: диссертация на соискание ученой степени кандидата технических наук. – 2005. – 115 с.
9. Мухачева Э. А., Мухачева А. С Л. В. Канторович и задачи раскроя-упаковки: новые подходы для решения комбинаторных задач линейного раскроя и прямоугольной упаковки: Записки научных семинаров ПОМИ Том 312. – 2004. – 255 с.
10. Мухачева Э. А., Назаров Д. А. Конструирование прямоугольных упаковок: алгоритм «перестройки» на базе блочных структур: Автоматика и телемеханика, №2. – 2008. – 113 с.
11. Картак В. М. Метод группировки для решения непрерывной задачи линейного раскроя: Дискретный анализ и исследование операций, май-июнь 2009 Том 16, №3. – 2009. – 47-62 с.
12. Руднев А. С. Алгоритм имитации отжига для решения задач двумерной прямоугольной упаковки в контейнеры с запрещенными областями: Дискретный анализ и исследование операций, июль-август 2010 Том 17, №4. – 2010. – 43-66 с.
13. Мухачева Э. А. Рациональный раскрой промышленных материалов. Применение АСУ. – М: «Машиностроение», 1984. – 176 с.
14. Канторович Л. В., Залгаллер В. А. Рациональный раскрой промышленных материалов. Издание второе, исправленное и дополненное. – М: «Наука», 1971. – 299 с.
15. Филлипова А. С. Обзор методов решения задач раскроя-упаковки уфимской научной школы Э. А. Мухачевой. – УГАТУ, 2011. – 27 с.
16. Мухавчева А. С., Смагин М. А. Гибридные алгоритмы для задачи прямоугольной упаковки на базе генетического метода «нижний левый» и «последовательного уточнения оценок». – Электрон. дан. – Режим доступа: smagin.on.ufanet.ru/Files/4.pdf
17. Задача прямоугольной упаковки: гибридизация классического генетического алгоритма с процедурами последовательного уточнения оценок». – Электрон. дан. – Режим доступа: smagin.on.ufanet.ru/Files/5.pdf
18. Э. А. Мухачева, А. В. Чиглинцев, М. А. Смагин, Э. А. Мухачева Задачи двумерной упаковки: развитие генетических алгоритмов на базе смешенных процедур локального поиска оптимального решения). – Электрон. дан. – Режим доступа: smagin.on.ufanet.ru/Files/6.pdf
19. Мухачева А. С., Чиглинцев А. В., Смагин М. А., Бунаков Д. И. Задача прямоугольной упаковки: гибридный алгоритм на базе генетических процедур (MChS). – Электрон. дан. – Режим доступа: smagin.on.ufanet.ru/Files/7.pdf
20. Э. А. Мухачева, А. С. Мухачева, М. А. Смагин Задача прямоугольной упаковки: классический генетический метод с алгоритмами «нижний левый» и «последовательного уточнения оценок». – Электрон. дан. – Режим доступа: smagin.on.ufanet.ru/Files/8.pdf
21. А. С. Мухачева, М. А. Смагин, И. П. Владимиров Задачи двумерной прямоугольной упаковки: генетический классический алгоритм на базе различных процедур расшифровки решений). – Электрон. дан. – Режим доступа: smagin.on.ufanet.ru/Files/10.pdf
22. Э. А. Мухачева, А. С. Мухачева, М. А. Смагин Методика тестирования и анализ результатов работы блочного и двойственного декодеров на базе генетического алгоритма). – Электрон. дан. – Режим доступа: smagin.on.ufanet.ru/Files/14.pdf
23. М. А. Смагин Автоматизированная система двумерной упаковки с использованием различных декодеров на базе генетического алгоритма). – Электрон. дан. – Режим доступа: smagin.on.ufanet.ru/Files/17.pdf
24. Э. А. Мухачева, А. Ф. Валеева, М. Н. Аглиуллин, М. А. Смагин, М. А. Месягутов Задача двумерной упаковки деталей в листы: алгоритмы «муравьиной колонии» и «наивный» алгоритм локального поиска с замещением). – Электрон. дан. – Режим доступа: smagin.on.ufanet.ru/Files/18.pdf

Приложение А. Текст программы

Form1

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Data.Sql;

using System.Data.SqlClient;

using System.Data.SqlServerCe;

using System.Data.SqlTypes;

using System.Data.Common;

using System.Drawing.Drawing2D;

using System.Runtime.InteropServices;

using Microsoft.SqlServer;

using Microsoft.SqlServer.Server;

using Tao.OpenGl;

using Tao.FreeGlut;

using Tao.Platform.Windows;

namespace WindowsFormsApplication1

{public partial class Form1 : Form

{ public Form1()

{InitializeComponent();

AnT.InitializeContexts();}

private void Form1\_Load(object sender, EventArgs e)

{// TODO: данная строка кода позволяет загрузить данные в таблицу "\_db\_DataSet.Текущий\_заказ". При необходимости она может быть перемещена или удалена.

this.текущий\_заказTableAdapter.Fill(this.\_db\_DataSet.Текущий\_заказ);

// TODO: данная строка кода позволяет загрузить данные в таблицу "\_db\_DataSet.Список\_материалов". При необходимости она может быть перемещена или удалена.

this.список\_материаловTableAdapter.Fill(this.\_db\_DataSet.Список\_материалов);

// TODO: данная строка кода позволяет загрузить данные в таблицу "\_db\_DataSet.Список\_изделий". При необходимости она может быть перемещена или удалена.

this.список\_изделийTableAdapter.Fill(this.\_db\_DataSet.Список\_изделий);

// TODO: данная строка кода позволяет загрузить данные в таблицу "\_db\_DataSet.Заказы". При необходимости она может быть перемещена или удалена.

this.заказыTableAdapter.Fill(this.\_db\_DataSet.Заказы);

// TODO: данная строка кода позволяет загрузить данные в таблицу "\_db\_DataSet.Заказ". При необходимости она может быть перемещена или удалена.

this.заказTableAdapter.Fill(this.\_db\_DataSet.Заказ);

// инициализация Glut

Glut.glutInit();

Glut.glutInitDisplayMode(Glut.GLUT\_RGB | Glut.GLUT\_DOUBLE | Glut.GLUT\_DEPTH);

// очитка окна

Gl.glClearColor(255, 255, 255, 1);

// установка порта вывода в соотвествии с размерами элемента anT

Gl.glViewport(0, 0, AnT.Width, AnT.Height);

// настройка проекции

Gl.glMatrixMode(Gl.GL\_PROJECTION);

Gl.glLoadIdentity();

// теперь необходимо корректно настроить 2D ортогональную проекцию

// в зависимости от того, какая сторона больше

// мы немного варьируем то, как будет сконфигурированный настройки проекции

if ((float)AnT.Width <= (float)AnT.Height)

{Glu.gluOrtho2D(0.0, 30.0 \* (float)AnT.Height / (float)AnT.Width, 0.0, 30.0); }

Else{Glu.gluOrtho2D(0.0, 30.0 \* (float)AnT.Width / (float)AnT.Height, 0.0, 30.0); }

Gl.glMatrixMode(Gl.GL\_MODELVIEW);

Gl.glLoadIdentity();

for (Int32 i = \_db\_DataSet.Заказ.Count; i > 0; i--)

{ \_db\_DataSet.Заказ[i-1].Delete(); }

заказTableAdapter.Update(\_db\_DataSet.Заказ);

\_db\_DataSet.Заказ.AcceptChanges();

dataGridView2.Refresh();}

//Добавление нового изделия в составляемый заказ

private void Dob\_v\_z\_i\_Click(object sender, EventArgs e)

{ for (Int32 i = 0; i < \_db\_DataSet.Список\_изделий.Rows.Count; i++)

{ if (dataGridView1.Rows[i].Cells[0].Value != null && dataGridView1.Rows[i].Cells[0].Value.ToString() != "False")

{Int32 fl = 1;

for (Int32 ij = 0; ij < \_db\_DataSet.Заказ.Rows.Count; ij++)

{ if (\_db\_DataSet.Заказ[ij][0].ToString() == \_db\_DataSet.Список\_изделий[i][0].ToString())

{ fl = 0; } }

if (fl == 1)

{ Int32 bb;

String cc, dd;

Int32.TryParse(\_db\_DataSet.Список\_изделий[i][0].ToString(), out bb);

cc = \_db\_DataSet.Список\_изделий[i][1].ToString();

dd = \_db\_DataSet.Список\_изделий[i][2].ToString();

заказTableAdapter.Insert(bb, cc, dd, 1);

заказTableAdapter.Update(\_db\_DataSet.Заказ);

заказTableAdapter.Fill(\_db\_DataSet.Заказ);

dataGridView2.Refresh(); }} dataGridView1.Rows[i].Cells[0].Value = null; }

for (Int32 i = 0; i < dataGridView1.RowCount; i++)

{ dataGridView1.Rows[i].Cells[0].Value = null; }

dataGridView2.Refresh();}

//Добавление нового материала в составляемый заказ

private void Dob\_v\_z\_m\_Click(object sender, EventArgs e)

{ Int32 sss;

Int32.TryParse(comboBox5.SelectedIndex.ToString(), out sss);

Int32 fl = 1;

for (Int32 ij = 0; ij < \_db\_DataSet.Заказ.Rows.Count; ij++)

{ if (\_db\_DataSet.Заказ[ij][0].ToString() == \_db\_DataSet.Список\_материалов[sss][0].ToString())

{fl = 0; } }

if (fl == 1)

{ Int32 bb;

String cc, dd;

Int32.TryParse(\_db\_DataSet.Список\_материалов[sss][0].ToString(), out bb);

cc = \_db\_DataSet.Список\_материалов[sss][1].ToString();

dd = \_db\_DataSet.Список\_материалов[sss][2].ToString();

заказTableAdapter.Insert(bb, cc, dd, 1);

заказTableAdapter.Update(\_db\_DataSet.Заказ);

заказTableAdapter.Fill(\_db\_DataSet.Заказ);

dataGridView2.Refresh(); } }

//заказы //Сохранение составленного заказа

private void Dob\_z\_s\_Click(object sender, EventArgs e)

{\_db\_DataSet.Заказы.AcceptChanges();

\_db\_DataSet.Заказ.AcceptChanges();

Int32 z = 0, zz = 0;

if (textBox2.Text == "3\*\*\*\*") { zz = 1; }

for (Int32 i = 0; i < \_db\_DataSet.Заказы.Rows.Count; i++)

{ if (\_db\_DataSet.Заказы[i][0].ToString() == textBox2.Text) { z = 1; } }

if (textBox1.Text == "" || zz == 1 || z == 1)

{ MessageBox.Show("Данные не корректны", "Ошибка ввода данных");}

else

{ Int32 bb;

String cc, dd = "";

Int32.TryParse(textBox2.Text, out bb);

cc = textBox1.Text;

for (Int32 i = 0; i < \_db\_DataSet.Заказ.Rows.Count; i++)

{dd += \_db\_DataSet.Заказ[i][0] + "x" + \_db\_DataSet.Заказ[i][3] + ";";}

//заказыTableAdapter.Insert(bb, cc, dd);

DataRow newrow = \_db\_DataSet.Заказы.NewRow();

newrow[0] = bb;

newrow[1] = cc;

newrow[2] = dd;

\_db\_DataSet.Заказы.Rows.Add(newrow);

заказыTableAdapter.Update(\_db\_DataSet.Заказы);

\_db\_DataSet.Заказы.AcceptChanges();

comboBox1.Refresh();

dataGridView3.Refresh(); }

for (Int32 i = \_db\_DataSet.Заказ.Count; i > 0; i--)

{ \_db\_DataSet.Заказ[i - 1].Delete(); }

заказTableAdapter.Update(\_db\_DataSet.Заказ);

\_db\_DataSet.Заказ.AcceptChanges();

dataGridView2.Refresh(); }

//заказы //udalenie zakaza

private void ud\_z(object sender, EventArgs e)

{ Int32 ii;

Int32.TryParse(comboBox1.Text.ToString(), out ii);

for (int i = 0; \_db\_DataSet.Заказы.Count > i; i++)

{ if (\_db\_DataSet.Заказы[i].Код\_заказа == ii)

{ \_db\_DataSet.Заказы.AcceptChanges();

\_db\_DataSet.Заказы.Rows[i].Delete();

i = \_db\_DataSet.Заказы.Count;} }

заказыTableAdapter.Update(\_db\_DataSet.Заказы);

\_db\_DataSet.Заказы.AcceptChanges();

comboBox1.Refresh();

dataGridView3.Refresh();

tek\_z(); }

private void tabControl1\_Click(object sender, EventArgs e)

{if (tabControl1.SelectedIndex == 1) { tek\_z(); } }

private void n\_tz()

{ for (Int32 i = \_db\_DataSet.Текущий\_заказ.Count; i > 0; i--)

{ \_db\_DataSet.Текущий\_заказ[i-1].Delete(); }

текущий\_заказTableAdapter.Update(\_db\_DataSet.Текущий\_заказ);

текущий\_заказTableAdapter.Fill(\_db\_DataSet.Текущий\_заказ);

dataGridView3.Refresh(); }

private void tek\_z()

{ n\_tz();

//dataGridView3.RowCount = 1;

Int32 ind;

Int32.TryParse(comboBox1.SelectedIndex.ToString(), out ind);

String ss, r = "", n = "";

ss = \_db\_DataSet.Заказы[ind][2].ToString();//набор

String[] sss = ss.Split(new char[] { ';' }, StringSplitOptions.RemoveEmptyEntries);

Int32 ttt = sss.Count();

for (Int32 i = 0; i < sss.Count(); i++)

{ String[] sss1 = sss[i].Split(new char[] { 'x' }, StringSplitOptions.RemoveEmptyEntries);

Int32 k, kl;

Int32.TryParse(sss1[0], out k);

Int32.TryParse(sss1[1], out kl);

for (Int32 j = 0; j < \_db\_DataSet.Список\_изделий.Rows.Count; j++)

{ Int32 kk;

Int32.TryParse(\_db\_DataSet.Список\_изделий[j][0].ToString(), out kk);

if (kk == k)

{ r = \_db\_DataSet.Список\_изделий[j][2].ToString();

n = \_db\_DataSet.Список\_изделий[j][1].ToString();

текущий\_заказTableAdapter.Insert(k, n, r, kl);

текущий\_заказTableAdapter.Update(\_db\_DataSet.Текущий\_заказ);

текущий\_заказTableAdapter.Fill(\_db\_DataSet.Текущий\_заказ);

//\_db\_DataSet.Текущий\_заказ.AcceptChanges();

dataGridView3.Refresh(); } }

for (Int32 j = 0; j < \_db\_DataSet.Список\_материалов.Rows.Count; j++)

{ Int32 kk;

Int32.TryParse(\_db\_DataSet.Список\_материалов[j][0].ToString(), out kk);

if (kk == k)

{ r = \_db\_DataSet.Список\_материалов[j][2].ToString();

n = \_db\_DataSet.Список\_материалов[j][1].ToString();

текущий\_заказTableAdapter.Insert(k, n, r, kl);

текущий\_заказTableAdapter.Update(\_db\_DataSet.Текущий\_заказ);

текущий\_заказTableAdapter.Fill(\_db\_DataSet.Текущий\_заказ);

dataGridView3.Refresh(); }

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{ tek\_z }

private void button2\_Click(object sender, EventArgs e)

{ tabControl1.SelectedIndex = 0; }

private void list(List<orthogonalI> ort, List<orthogonalI> material)

{ Int32 ind, mx = 0, my = 0, H = 0, L = 0;

Int32.TryParse(comboBox1.SelectedIndex.ToString(), out ind);

String ss, r = "";

ss = \_db\_DataSet.Заказы[ind][2].ToString();//набор

String[] sss = ss.Split(new char[] { ';' }, StringSplitOptions.RemoveEmptyEntries);

Int32 ttt = sss.Count();

//orthogonalI []ortho; new orthogonalI();

for (Int32 i = 0; i < sss.Count(); i++)

{ String[] sss1 = sss[i].Split(new char[] { 'x' }, StringSplitOptions.RemoveEmptyEntries);

Int32 k, kl;

Int32.TryParse(sss1[0], out k);

Int32.TryParse(sss1[1], out kl);

for (Int32 j = 0; j < \_db\_DataSet.Список\_изделий.Rows.Count; j++)

{ Int32 kk;

Int32.TryParse(\_db\_DataSet.Список\_изделий[j][0].ToString(), out kk);

if (kk == k)

{ r = \_db\_DataSet.Список\_изделий[j][2].ToString();

String[] rr = r.Split(new char[] { 'x' }, StringSplitOptions.RemoveEmptyEntries);

Int32.TryParse(rr[0], out L);

Int32.TryParse(rr[1], out H);

for (Int32 ll = 0; ll < kl; ll++)

{ orthogonalI ortho = new orthogonalI();

ortho.oX = -1;

ortho.oY = -1;

ortho.H = -1;

ortho.L = -1;

ortho.L = L;

ortho.H = H;

ort.Add(ortho); } } }

for (Int32 j = 0; j < \_db\_DataSet.Список\_материалов.Rows.Count; j++)

{ Int32 kk;

Int32.TryParse(\_db\_DataSet.Список\_материалов[j][0].ToString(), out kk);

if (kk == k)

{ r = \_db\_DataSet.Список\_материалов[j][2].ToString();

String[] rr = r.Split(new char[] { 'x' }, StringSplitOptions.RemoveEmptyEntries);

Int32.TryParse(rr[0], out mx);

Int32.TryParse(rr[1], out my);

orthogonalI ortho = new orthogonalI();

ortho.oX = -1;

ortho.oY = -1;

ortho.H = -1;

ortho.L = -1;

ortho.H = my;

ortho.L = mx;

ortho.oX = 0;

ortho.oY = 0;

material.Insert(material.Count, ortho); } } } }

private void Raschet\_Click(object sender, EventArgs e)

{ tabControl1.SelectedIndex = 2;

// очищаем буфер цвета

Gl.glClear(Gl.GL\_COLOR\_BUFFER\_BIT);

// очищаем текущую матрицу

Gl.glLoadIdentity();

// устанавливаем текущий цвет - красный

Gl.glColor3f(0, 0, 0);

List<orthogonalI> ortbas = new List<orthogonalI>();

List<orthogonalI> mat = new List<orthogonalI>();

list(ortbas, mat);

Gl.glColor3f(0, 255, 0);

mat[0].Draw();

Gl.glColor3f(0, 0, 0);

Gl.glEnd();

Gl.glColor3f(0, 255, 0);

Gl.glBegin(Gl.GL\_TRIANGLES);

Gl.glVertex2d(0, mat[0].H);

Gl.glVertex2d(mat[0].L, mat[0].H);

Gl.glVertex2d(mat[0].L, 0);

Gl.glEnd();

Gl.glBegin(Gl.GL\_TRIANGLES);

Gl.glVertex2d(0, mat[0].H);

Gl.glVertex2d(0, 0);

Gl.glVertex2d(mat[0].L, 0);

Gl.glEnd();

Gl.glColor3f(0, 0, 0);

if (comboBox3.Text == "Генетический" || comboBox3.Text == "Левый нижний")

{ List<orthogonalI> ort = new List<orthogonalI>();

if (comboBox3.Text == "Генетический")

{ List<int> osob = new List<int>();

osob=ga(ortbas, mat);

for (int i = 0; i < osob.Count; i++)

{ if (osob[i] > 0)//esli bez povorota

{ ort.Add(ortbas[osob[i] - 1]); }

else//esli s povorotom

{orthogonalI onew = new orthogonalI();

onew.H = ortbas[0 - (osob[i] + 1)].L;

onew.L = ortbas[0 - (osob[i] + 1)].H;

ort.Add(onew); } } }

else { ort = ortbas; }

Int32 klkl = 0;

for (Int32 i = 0; i < ort.Count(); i++) { ort[i].oX = -1; ort[i].oY = -1; }

for (Int32 k = 0; k < ort.Count(); k++)

{ ort[k].oX = mat[0].L - ort[k].L;

ort[k].oY = mat[0].H - ort[k].H;

for (Int32 i = ort[k].oY; i > -1; i--)//по y

{ for (Int32 j = ort[k].oX; j > -1; j--)//по x

{ Int32 p = 0;

for (Int32 l = k - 1; l > -1; l--)//по y

{ if (!(j >= ort[l].oX + ort[l].L || i >= ort[l].oY + ort[l].H || j + ort[k].L <= ort[l].oX || i + ort[k].H <= ort[l].oY))

{ p = 1;

if (i == ort[k].oY && j == ort[k].oX)

{ klkl = 1; } } }

if (p == 0)

{ ort[k].oX = j;

ort[k].oY = i; } } }

if (klkl == 0)

{ ort[k].zalivka();

ort[k].Draw(); }} }

// дожидаемся конца визуализации кадра

Gl.glFlush();

//for (Int32; ; )

// посылаем сигнал перерисовки элемента AnT.

AnT.Invalidate();}

//geneticheskii algoritm

List<int> ga(List<orthogonalI> ort, List<orthogonalI> mat)

{//sozdaem 1e pokolenie

List<List<int>> pop=new List<List<int>>();

List<int> svc=new List<int>();

//prohodim po osobiam

for (int i = 0; i < 10; i++)

{ pop.Add(new List<int>());

//prohodim v osobi po genam

for (int j = 0; j < ort.Count; )

{ int rr;

//sozdaem sluchainii gen

rr = rndCount(ort.Count); //r.Next(-ort.Count, ort.Count);

int f = 0;

//proverka na vhojdenie poluchennogo gena v dannuiu osob

if (j!=0)

{ for (int k = 0; k < pop[i].Count; k++)

{//iskliuchenie nulia, chto bi dalee mojno bilo ispolzovat otricatelnie znacheniia

if (pop[i][k] == rr || rr == 0 || pop[i][k] == -rr)

{ f = 1; } } }

//esli net vhojdeniia to dobavliaem

if(f==0&&rr!=0)

{ if (rr < 0 )

{ if (ort[(rr + 1) \* (-1)].H == ort[(rr + 1) \* (-1)].L)

{ rr \*= -1; }}

pop[i].Add(rr);

j++; } }

int ff = 0;

//proverka na vhogdenie poluchennoi osobi v pokolenie

for (int k = 0; k < pop.Count; k++)

{if (pop[i] == pop[k]&&i!=k)

{ ff = 1;

pop.RemoveAt(i); }}

if (ff == 0)

{ svc.Add(ocenka(ort, mat, pop[i])); }

}//sozdali pokolenie

int par1 = 0, par2 = 0,svcl = 0;

//List<int> child1 = new List<int>(), child2 = new List<int>();

for (int i = 0; i < 100; i++)//prohodim evoluciyu do 100 pokolenia

{ //vibiraem roditelei

par(pop, svc, par1, par2);

//skreshivaem

cros(pop, par1, par2, ort, mat, svc);

//mutiruem

mut(pop, ort, mat, svc);

//sokrashenie populiacii do razmera pervonachalnoi

for (; pop.Count != 10; )

{ pop.RemoveAt(svc.IndexOf(svc.Min()));

svc.RemoveAt(svc.IndexOf(svc.Min())); } }

//vibor luchshei ocenki

svcl = 0;

for (int i = 1; i < svc.Count; i++)

{ if (svc[svcl] < svc[i])

{ svcl = i; } }

return pop[svcl]; }

//vibor roditelei //vozvrashaet indeksi roditelei iz populiacii

void par(List<List<int>> pop, List<int> svc, int par1, int par2)

{//Random r = new Random(0);

int rr,rr1,ii;

for (int i = 0; i < 2; i++)

{ rr1 = 0;

ii = 0;

for (int j = 0; j < pop.Count; j++)

{ rr = rndCount2(0,10);//r.Next(0, 100);

if (rr+svc[j] > rr1+svc[ii])

{if (i == 0 || (i == 1 && par1 != ii))

{ rr1 = rr;

ii = j; } } }

if (i == 0) { par1 = ii;//rr1; }

else { par2 = ii;// rr1; }} }

//skreshivanie

void cros(List<List<int>> pop, int par1, int par2, List<orthogonalI> ort, List<orthogonalI> mat, List<int> svc)

{ List<int> child1 = new List<int>();

List<int> child2 = new List<int>();

//Random rp = new Random(0);

//Random rq = new Random(0);

int p=rndCount2(0, pop[0].Count - 2);//rp.Next(0, pop[0].Count - 2);

int q = rndCount2(p + 1, pop[0].Count - 1);//rq.Next(p+1, pop[0].Count - 1);

//sostavliaem nachalniie cepochki dlia child 1 i 2 iz par 1 i 2

for (int i = p; i <= q; i++)

{ child1.Add(pop[par1][i]);

child2.Add(pop[par2][i]); }

//dopolniaem child1 iz par2

for (int i = pop[par2].IndexOf(child1[child1.Count-1]); child1.Count() < pop[par2].Count && i < pop[par2].Count; i++)

{ if (child1.IndexOf(pop[par2][i]) < 0)

{ child1.Add(pop[par2][i]); } }

for (int i = 0; child1.Count() < pop[par2].Count && i < pop[par2].Count; i++)

{ if (child1.IndexOf(pop[par2][i]) < 0)

{ child1.Add(pop[par2][i]); } }

//dopolniaem child2 iz par1

for (int i = pop[par1].IndexOf(child2[child2.Count-1]); child2.Count() < pop[par1].Count && i < pop[par1].Count; i++)

{ if (child2.IndexOf(pop[par1][i]) < 0)

{ child2.Add(pop[par1][i]); } }

for (int i = 0; child2.Count() < pop[par1].Count && i < pop[par1].Count; i++)

{ if (child2.IndexOf(pop[par1][i]) < 0)

{ child2.Add(pop[par1][i]); }}

//dobavliaem v populiaciiu detei, kotorih v nei eshe net

if (pop.IndexOf(child1) < 0 || pop.IndexOf(child1) >= pop.Count)

{ pop.Add(child1);

svc.Add(ocenka(ort, mat, pop[pop.Count-1])); }

if (pop.IndexOf(child2) < 0 || pop.IndexOf(child2) >= pop.Count)

{ pop.Add(child2);

svc.Add(ocenka(ort, mat, pop[pop.Count-1])); } }

//mutaciya

void mut(List<List<int>> pop, List<orthogonalI> ort, List<orthogonalI> mat, List<int> svc)

{ //Random r = new Random(0);

for (int i = 0; i < pop.Count; i++)

{ int m;

m = rndCount2(0,100);//r.Next(0, 100);

//mutaciia perestanovki 2 genov

if (m < 30)

{ List<int> a = new List<int>();

for(int j=0;j<pop[i].Count;j++)

{ a.Add(pop[i][j]); }

int m1, m2;

m1 = rndCount2(0, pop[i].Count - 1);//r.Next(0,pop[i].Count-1);

m2 = m1;

for (; m1==m2; )

{ m2 = rndCount2(0, pop[i].Count - 1);//r.Next(0, pop[i].Count - 1); }

a[m1]=pop[i][m2];

a[m2]=pop[i][m1];

if (pop.IndexOf(a) < 0 || pop.IndexOf(a) >= pop.Count-1)

{ pop.Add(a);

svc.Add(ocenka(ort, mat, pop[pop.Count-1])); } }

int mm = rndCount2(0,100);//r.Next(0, 100);

int m3;

m3 = rndCount2(0, pop[i].Count - 1);//r.Next(0, pop[i].Count - 1);

int o = 1;

if (pop[i][m3] < 0) { o = -1; }

//mutaciia povorota gena

if (mm < 30 && ort[o \* pop[i][m3]-1].H != ort[o \* pop[i][m3]-1].L)

{ if (m < 30)

{ List<int> b = new List<int>();

for (int j = 0; j < pop[i].Count; j++)

{ b.Add(pop[pop.Count - 1][j]); }

b[m3] = -pop[pop.Count - 1][m3];

if (pop.IndexOf(b) < 0 || pop.IndexOf(b) >= pop.Count - 1)

{ pop.Add(b);

svc.Add(ocenka(ort, mat, pop[pop.Count - 1])); }}

List<int> a = new List<int>();

for (int j = 0; j < pop[i].Count; j++)

{ a.Add(pop[i][j]); }

a[m3] = -pop[i][m3];

if (pop.IndexOf(a) < 0 || pop.IndexOf(a) >= pop.Count-1)

{ pop.Add(a);

svc.Add(ocenka(ort, mat, pop[pop.Count-1])); }}} }

//ocenka raskroia

int ocenka(List<orthogonalI> ortbas, List<orthogonalI> mat, List<int> osob)

{//spisok detalei,material,raskroi

int ocenka;

List<orthogonalI> ort = new List<orthogonalI>();

//preobrazovanie posledovatelnosti v fenotip

for (int i = 0; i < osob.Count; i++)

{ if (osob[i] > 0)//esli bez povorota

{ort.Add(ortbas[osob[i] - 1]); }

else//esli s povorotom

{ orthogonalI onew = new orthogonalI();

//onew = ortbas[0 - (osob[i] + 1)];

onew.H = ortbas[0 - (osob[i] + 1)].L;

onew.L = ortbas[0 - (osob[i] + 1)].H;

ort.Add(onew); } }

//obnulenie koordinat

for (Int32 i = 0; i < ort.Count(); i++) { ort[i].oX = -1; ort[i].oY = -1; }

for (Int32 k = 0; k < ort.Count(); k++)

{ for (Int32 i = mat[0].H - ort[k].H; i > -1; i--)//по y

{ for (Int32 j = mat[0].L - ort[k].L; j > -1; j--)//по x

{ Int32 p = 0;

for (Int32 l = k - 1; l > -1; l--)//по y

{ if (!(j >= ort[l].oX + ort[l].L || i >= ort[l].oY + ort[l].H))

{ p = 1; } }

if (p == 0)

{ ort[k].oX = j;

ort[k].oY = i; } } } }

ocenka = 999999;

int otr = 0;

//ocenka po dline ostatka

for (int i = 0; i < ort.Count; i++)

{ if (ort[i].oY < 0 || ort[i].oX < 0)

{otr++; }

if (ort[i].oX + ort[i].L <= mat[0].L)

{ int asd = mat[0].L - (ort[i].oX + ort[i].L);

if (asd-(otr\*10) < ocenka && ort[i].oY >= 0 && ort[i].oX>=0)

{ ocenka = asd-(otr\*10); } } }

return ocenka; }

//svc

void ocenkasvc(List<orthogonalI> ortbas, List<orthogonalI> mat, List<int> osob)

{//spisok detalei,material,raskroi

List<orthogonalI> ort = new List<orthogonalI>();

//preobrazovanie posledovatelnosti v fenotip

for (int i = 0; i < osob.Count; i++)

{ if (osob[i] > 0)//esli bez povorota

{ort.Add(ortbas[osob[i] - 1]); }

else//esli s povorotom

{ orthogonalI onew = new orthogonalI();

//onew = ortbas[0 - (osob[i] + 1)];

onew.H = ortbas[0 - (osob[i] + 1)].L;

onew.L = ortbas[0 - (osob[i] + 1)].H;

ort.Add(onew); } }

//obnulenie koordinat

for (Int32 i = 0; i < ort.Count(); i++) { ort[i].oX = -1; ort[i].oY = -1; }

//ibl

for (Int32 k = 0; k < ort.Count(); k++)

{ort[k].oX = mat[0].L - ort[k].L;

ort[k].oY = mat[0].H - ort[k].H;

for (Int32 i = ort[k].oY; i > -1; i--)//по y

{ for (Int32 j = ort[k].oX; j > -1; j--)//по x

{ Int32 p = 0;

for (Int32 l = k - 1; l > -1; l--)//по y

{ //proverka na peresechenie s drugimi detaliami

if (!(j >= ort[l].oX + ort[l].L || i >= ort[l].oY + ort[l].H || j + ort[k].L <= ort[l].oX || i + ort[k].H <= ort[l].oY))

{ p = 1; } }

if (p == 0)

{ ort[k].oX = j;

ort[k].oY = i; } } }}

//svc

List<List<int>> blok=new List<List<int>>();

int b=-1;

int bb = 0;

//delim na bloki

for (int i = 0; i < mat[0].L; i++)

{ for (int j = 0; j < ort.Count; j++)

{ if (ort[j].L + ort[j].oX == i&&(b==-1||blok[b][0]!=i))

{ List<int> bl = new List<int>();

bl.Add(bb); bl.Add(i);

blok.Add(bl);

bb = i;

b++; } } }

List<List<int>> normaij = new List<List<int>>();

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

{List<int> n = new List<int>();

for (int j = 0; j < ort.Count; j++)

{n.Add(0);

if ((ort[j].oX >= blok[i][0] && ort[j].oX <= blok[i][1]) || (ort[j].oX + ort[j].L >= blok[i][0] && ort[j].oX + ort[j].L <= blok[i][1]))

{ n[n.Count-1] = ort[j].H \* mat[0].H; } }

normaij.Add(n); }

List<int> ly = new List<int>();

for (int i = 0; i <normaij.Count; i++)

{ ly.Add(mat[0].H);

for (int j = 0; j < normaij[i].Count; j++)

{ if (normaij[j][i] > 0)

{ ly[i] -= ort[j].H; } }}

string a = "";

for (int i = 0; i < normaij.Count; i++)

{ a += " " + ly[i] + " |"; }

MessageBox.Show("", a);

List<double> y0 = new List<double>();

for (int j = 0; j < ort.Count; j++)

{y0.Add(0);

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

{ y0[j]+=(normaij[i][j]\*(blok[i][1]-blok[i][0]))/(mat[0].H-ly[i]); }

y0[j]/=ort[j].H;

y0[j]/=ort[j].L; }

string aa = "";

for (int i = 0; i <ort.Count; i++)

{ aa+=" " + y0[i] + " |";}

MessageBox.Show("",aa); }

//random ot count do count2, neotricatelnii

int rndCount2(int count,int count2)

{ var buf = Guid.NewGuid().ToByteArray();

var randomValue32 = BitConverter.ToInt32(buf, 4);

int u = count2 - count + 1;

var val = randomValue32 % u;

val += count;

return val; }

int rndCount(int count)

{ var buf = Guid.NewGuid().ToByteArray();

var randomValue32 = BitConverter.ToInt32(buf, 4);

var val = randomValue32 % (2\*count+1);

if (val > count) val = count - val;

return val; } }}

Класс OrthogonalI

class orthogonalI:Item

{public Int32 oX, oY;//, L, H;

public Int32 L { get; set; }

public Int32 H { get; set; }

public void Draw()

{ Gl.glBegin(Gl.GL\_LINE\_LOOP);

Gl.glVertex2d(oX, oY);

Gl.glVertex2d(oX+L, oY);

Gl.glVertex2d(oX+L, oY+H);

Gl.glVertex2d(oX, oY+H);

//Gl.glVertex2d(oX, oY);

Gl.glEnd();}

public void zalivka()

{Gl.glColor3f(0, 255, 255);

Gl.glBegin(Gl.GL\_TRIANGLES);

Gl.glVertex2d(oX, oY + H);

Gl.glVertex2d(oX + L, oY + H);

Gl.glVertex2d(oX + L, oY);

Gl.glEnd();

Gl.glBegin(Gl.GL\_TRIANGLES);

Gl.glVertex2d(oX, oY + H);

Gl.glVertex2d(oX, oY);

Gl.glVertex2d(oX + L, oY);

Gl.glEnd();

Gl.glColor3f(0, 0, 0); } }