Вестник Сыктывкарского университета.
Сер.1. Вып.10. 2009

УДК 004.428

ПЛАТФОРМА ДЛЯ ТЕСТИРОВАНИЯ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ ФОРМАТНОГО РАСКРОЯ

А.Е. Подоров, Д.Ю. Саковнич

Разработаны общие принципы построения приложений по работе с ЗФР. Основываясь на этих принципах, была разработана платформа для тестирования алгоритмов решения ЗФР. Приведены результаты применения платформы для сравнения разработанных алгоритмов.

1. Введение

В настоящее время разработано достаточно большое количество алгоритмов и программных комплексов, основанных на них. Все эти программные продукты достаточно сильно различаются, начиная от данных, с которыми они оперируют, заканчивая графическим интерфейсом приложения.

Для работы с данными программами пользователю необходимы исходные данные задач ЗФР, в которые наряду с типичными для этих задач портфелями заказов, может входить и специфическая информация, присущая конкретному алгоритму, будь то данные, управляющие поведением алгоритма или ограничения по вырожденности полученного решения. В зависимости от этого могут различаться способы хранения и структура исходных данных, которые могут храниться в различных текстовых файлах, в таблицах базы данных или располагаться на каком-либо web–сервере предприятия. Несмотря на то, что входные данные неоднородны, они семантически схожи, поэтому существует возможность их унифицировать. Для этих целей можно разработать модуль, который бы поддерживал различные форматы входных данных и позволял преобразовывать их к общему внутреннему представлению.

Далее, наряду с неоднородностью исходных данных существует еще одна проблема — это различные для каждой программы представление выходных данных. К примеру, у пользователя имеется несколько программ решения ЗФР, ему нужно просмотреть и проанализировать результаты. Во-первых, если он уже привык и приобрел навыки манипуляции и анализа выходных данных, полученных конкретным приложением, ему уже будет сложно перестроиться на другое, которое может выдавать решение в совершенно другом формате. Мало того, что этот процесс перехода на новое приложение весьма трудоемкий, но к тому же чреват ошибками, которые порой трудно обнаружить. Во-вторых, возникает проблема сравнения результатов, отображаемых различными оболочками.

Алгоритмы и программы для решения ЗФР пишутся разными разработчиками, иногда бывает, что один и тот же алгоритм пишется разными разработчиками, каждый из них имеет свой образ мышления представление задачи и квалификацию. Соответственно, из-за этого алгоритмы получались несовместимыми между собой, а попытка их унифицировать приводила к неудаче. Эту неоднородность реализации призваны устранить общие входные и выходные данные.

Для того, чтобы протестировать и выявить поведение какого-либо алгоритма, необходимы тесты, которые представляют из себя набор входных данных. Если бы появились общие входные и выходные данные, то была бы ликвидирована неоднородность алгоритмов, и появились бы широкие возможности для тестирования.

Таким образом, возникла идея создания платформы (приложения), позволяющее работать со всеми разрабатываемыми алгоритмами и проводить их тестирование, она позволила бы управлять подключенными к ней алгоритмами, а также отображать результаты их работы, проводить исследование эффективности различных алгоритмов.

2. Стандартизация поведения алгоритмов

Для того, чтобы иметь возможность тестировать уже имеющиеся, а также будущие реализации алгоритмов, необходимо зафиксировать (стандартизировать) поведение всех разрабатываемых программ. Рассмотрим в целом все алгоритмы и выделим все то, что их объединяет.

На данный момент исследуется несколько модификаций задачи форматного раскроя (ЗФР): задача с фиксированным вектором требований (ВТ) [1,3], задача с люфтом входных данных (нежестко заданный ВТ), а также задача оптимального раскроя отходов сырья [2]. Все эти задачи требуют специфических входных данных. Например, в задаче с

фиксированным ВТ имеются жестко заданные ограничения на выкраиваемые заготовки, а также считается, что раскраивается материал одного размера. В этой задаче нас интересуют способы раскроя исходного материала и их интенсивности. В то же время в задаче с люфтом на вход помимо ВТ подаются его отклонения, как в большую, так и в меньшую сторону. В задаче оптимального раскроя материалов мы уже имеем дело не только с ограничениями, накладываемыми на количество выкраиваемых заготовок, но и с ограничениями на количество раскраиваемого материала. При этом считается, что имеется несколько материалов с различными характеристиками и размерами. Решением последней задачи являются способы раскроя **конкретного материала** с соответствующими интенсивностями, что необходимо учитывать при формировании решения.

Несмотря на все различия исследуемых задач, у них имеется довольно много общего и с некоторыми допущениями их можно унифицировать.

2.1. входные данные

Как можно заметить, все задачи имеют дело с двумя фундаментальными в рамках ЗФР понятиями. Это исходное сырье и его раскройные характеристики (далее материал), и вырезаемые заготовки (далее форматы). Для них выделено два соответствующих класса. Их исходный код приведен ниже.

```
public class Format {
  private int length; //размер
  private int diameter; //диаметр
  private int deltaP=0; //отклонение "+"
  private int deltaM=0; //отклонение "-"
  ......
}
 public class Material {
  private int length;//размеры
  private int knifeCount;//количество ножей на ПРС
  private int maxEdge; //максимальная кромка
  private int minEdge = 0; //минимальная кромка
  private int knifeInterval; //мин. расст. между ножами
  ......
}
```

Все исходные коды приведены на языке программирования java и взяты из реальной программы. При этом были отброшены только несущественные моменты.

Далее, анализируя все задачи, можно заметить, что каждая из них принимает на вход вектор требований, который является списком, хранящим форматы и соответствующие количества, которые необходимо вырезать. Помимо ВТ всем задачам необходима информация о раскраиваемом материале. При этом есть ряд задач, которые раскраивают не бесконечное, а ограниченное количество материалов, причем характеристики материалов в задаче могут различаться. Таким задачам на вход подается список материалов и их количество, которое можно использовать для раскроя. Этим требованием такие алгоритмы довольно сильно отличаются от тех, которые имеют дело с одним, причем в бесконечном количестве, материалом. Поэтому, чтобы двигаться дальше, все алгоритмы были обобщены таким образом, чтобы они все брали на вход помимо ВТ еще и список материалов и их количество. В случае если алгоритм не требует более одного материала, то он будет использовать первый материал из списка. Таким образом, введя два дополнительных объекта, хранящих пары <формат, количество>, <материал, количество>, мы пришли к следующей схеме входных данных:

```
//класс хранящий пару <формат, количество>
public class FormatCount {
  private Format format;//формат
  private int count;//количество
  ......
}
//класс хранящий пару <материал,количество>
public class MaterialCount {
  private Material material;//материал
  private int count;//количество
  ......
}
//класс входных данных алгоритмов
public class InputData {
    //используемый набор материалов
    private List<MaterialCount> materialCountList;
    //вектор требований
    private List<FormatCount> formatCountList;
    ......
}
```

2.2. выходные данные

Рассмотрим теперь выходные данные алгоритмов. Тут дело обстоит несколько проще, так как все они решают, по сути, одну и ту же задачу, а именно задачу раскроя материла на форматы, и результатом

их работы всегда является план раскроя. Единственное, что требуется здесь, так это четко определить ту структуру результата, которую будут возвращать все алгоритмы.

Решением задачи форматного раскроя является набор способов раскроя и их интенсивностей. Способ раскроя, в свою очередь, является упорядоченным списком вырезаемых форматов из конкретного материала, возможно с указанием позиции формата. Эта идея выходных данных реализована в следующих классах.

```
//класс решения ЗФР
public class TrimResult extends ArrayList<Pattern> {
   public int getTail()  {...} //общий отход решения
   public int getUsedArea() {...} //общая используемая площадь
   public double getEfficiency() {...} //эффективность
  .......
}
//способ раскроя и  интенсивность его использования
public class Pattern extends ArrayList<PatternItem> {
  //раскраиваемый материал
  private Material material;
  //интенсивность способа раскроя
  private int count;
  ......
}
// "обертка" для выкраиваемого формата
public class PatternItem {
  private Format format;
  .......
}
```

Класс PatternItem создан для того, чтобы при необходимости была возможность добавить некоторую дополнительную информацию к формату, выкраиваемому на данном конкретном материале.

2.3. поведение алгоритмов

Чтобы иметь возможность разрабатывать гибкие средства для тестирования или иного использования алгоритмов, необходимо, чтобы все реализации алгоритмов имели схожее поведение. Это позволило бы запускать тот или иной алгоритм, совершенно ничего «не зная» о нем.

Эта идея реализована с помощью введения базового класса Trim с абстрактным методом solve, запускающим расчеты.

```
//Базовый класс для всех алгоритмов
public abstract class Trim {
```

```
  //Запуск алгоритма решения
  public abstract TrimResult solve(InputData inputData);
  //Название алгоритма
  public abstract String getName();
  //Версия алгоритма
  public abstract String getVersion();
  //Описание алгоритма
  public abstract String getDescription();
}
```

Все классы–алгоритмы, унаследованные от Trim, получают стандартное поведение алгоритмов решения ЗФР.

3. Разработка платформы для тестирования

Как уже говорилось ранее, имеется ряд определенных трудностей, связанных с тестированием разрабатываемых алгоритмов. В основном все они связаны с отсутствием пользовательского интерфейса у разрабатываемых и даже некоторых реализованных алгоритмов. Разработка же графической оболочки требует времени, причем качественный интерфейс может потребовать от разработчика даже больше времени, чем реализация алгоритмов, что, несомненно, отвлекает от сути исследований. Основываясь на принципах, описанных в п.2, была разработана платформа для тестирования различных реализаций алгоритмов решения ЗФР. При ее разработке учитывались следующие требования:

1. Приложение должно позволять проводить тестирования любых алгоритмов, удовлетворяющих стандарту поведения из п.2.

2. Должна иметься возможность добавлять в приложение новые алгоритмы без его перекомпиляции. Желательно даже без его перезапуска.

3. Приложение должно позволять управлять входными и выходными данными, а также давать возможность выбора запускаемых алгоритмов. То есть оно должно стать полноценной графической оболочкой для любого алгоритма.

4. Выгрузка полученных решений для дальнейшей обработки

5. Возможность автоматической генерации тестов и тестирование на них выбранных алгоритмов

Для реализации описанного приложения была выбрана платформа и язык программирования java, так как обладает несколькими существенными достоинствами. Наиболее важным из них является кроссплатформенность. Программа, написанная на java, будет одинаково работать как на платформе Ms Windows, так и под любой другой операционной системой, для которой

имеется виртуальная машина java, например Linux. Java также является бесплатной платформой, что позволяет в случае распространения программного продукта включить интерпретатор java в свой дистрибутив. Java как язык программирования является полностью объектно–ориентированным даже на этапе выполнения программы. Это позволяет вызывать методы объекта по их названию и пользоваться другими преимуществами механизма рефлексии (reflection).

На рисунке 1 представлено главное окно приложения.

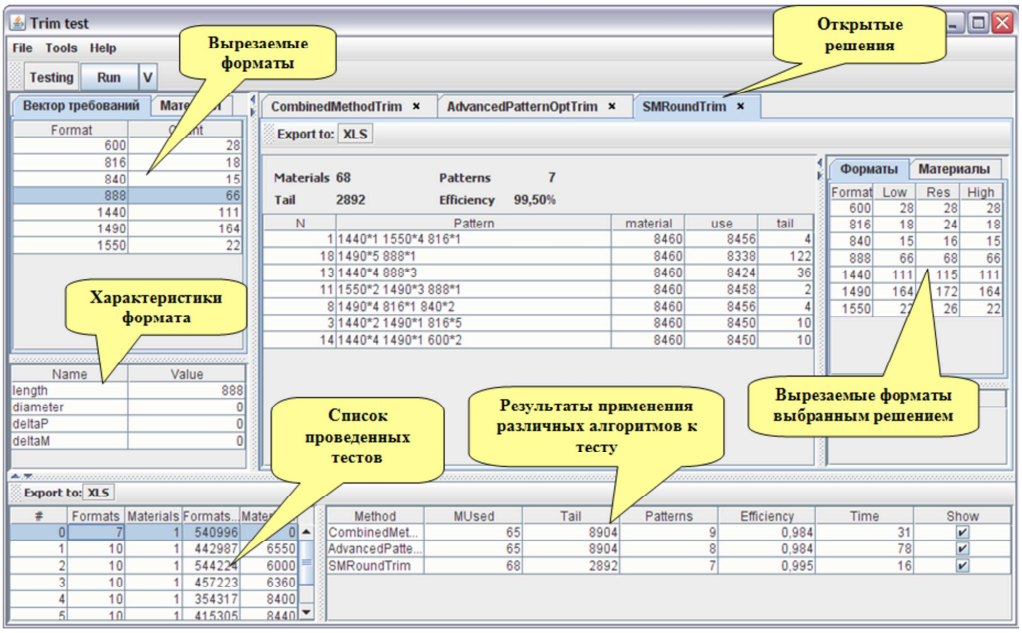
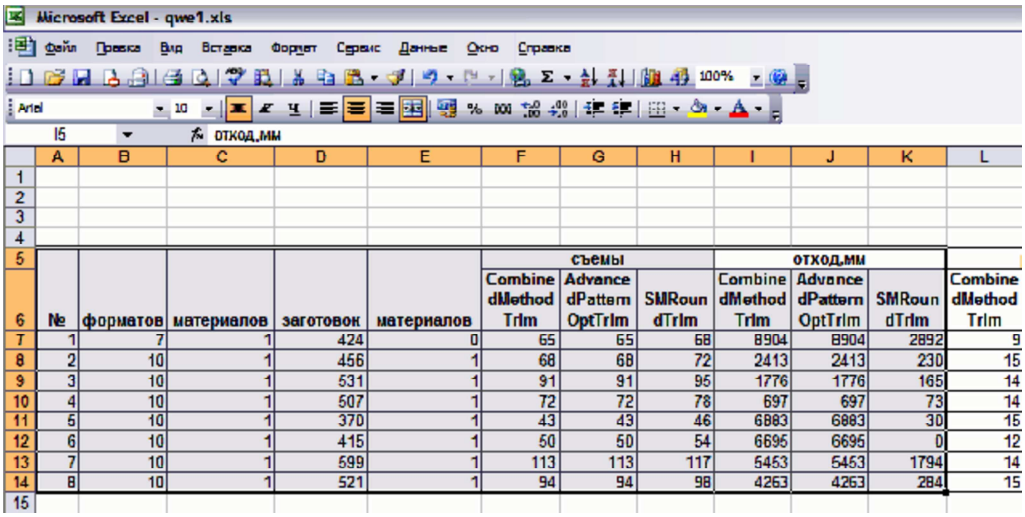


Рис. 1: Главное окно приложения

В рабочей области приложения мы попытались разместить как можно больше информации о характеристиках решений, чтобы была возможность сравнивать их, не прибегая для анализа к экспорту решений во внешние форматы. В случае проведения статистических исследований имеется возможность экспорта в таблицы Microsoft Excel.

Одной из основных функций приложения является проведения массового тестирования алгоритмов на тестах различного размера. Для этих целей был разработан модуль генерации тестов, позволяющий создавать набор тестов по заданным критериям, влияющим на размер и сложность решаемой задачи. К основным таким критериям относятся количество различных форматов в тесте и количество различных материалов (для алгоритмов, использующих на входе несколько различных материалов).



№	форматов	материалов	заготовок	материалов	съемы			отход,мм			
					CombinedMethodTrim	AdvancedPatternOptTrim	SMRoundTrim	CombinedMethodTrim	AdvancedPatternOptTrim	SMRoundTrim	CombinedMethodTrim
1	7	1	424	0	65	65	68	8904	8904	2892	9
2	10	1	456	1	68	68	72	2413	2413	230	15
3	10	1	531	1	91	91	95	1776	1776	165	14
4	10	1	507	1	72	72	78	697	697	73	14
5	10	1	370	1	43	43	46	6883	6883	30	15
6	10	1	415	1	50	50	54	6695	6695	0	12
7	10	1	599	1	113	113	117	5453	5453	1794	14
8	10	1	521	1	94	94	98	4263	4263	284	15

Рис. 2: Вид отчета, выгружаемого в Ms Excel

При проведении тестирования имеется возможность выбора тех алгоритмов, которые будут применены для каждого сгенерированного теста. Набор алгоритмов, с которыми может работать пользователь, заранее не фиксирован, и пользователь может добавить дополнительную группу алгоритмов, подключив соответствующий плагин. В роли плагина для данного приложения может выступать любой java–архив (*.jar файл), содержащий алгоритмы решения ЗФР (любой не абстрактный класс, унаследованный от класса trim.Trim). Так как динамическая загрузка и управление алгоритмами решения является достаточно важной, автономной и нетривиальной задачей, то для ее решения был реализован модуль управления плагинами, которым также можно воспользоваться при реализации других приложений по работе с ЗФР.

4. Применение платформы

С помощью разработанного приложения были протестированы некоторые уже имеющиеся алгоритмы.

Для сравнения различных алгоритмов решения ЗФР введем следующие критерии:

1. Эффективность — отношение суммарной длины всех форматов к суммарной длине всех используемых материалов (тамбуров), умноженное на 100 процентов.

2. Вырожденность — количество уникальных способов раскроя, что соответствует числу перенастроек режущего станка

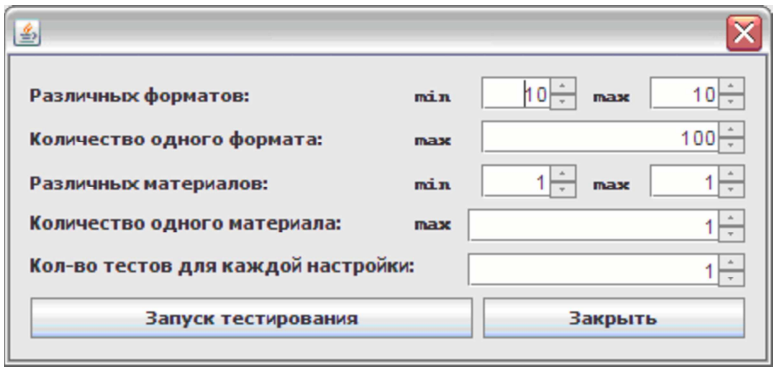


Рис. 3: Окно массового тестирования алгоритмов

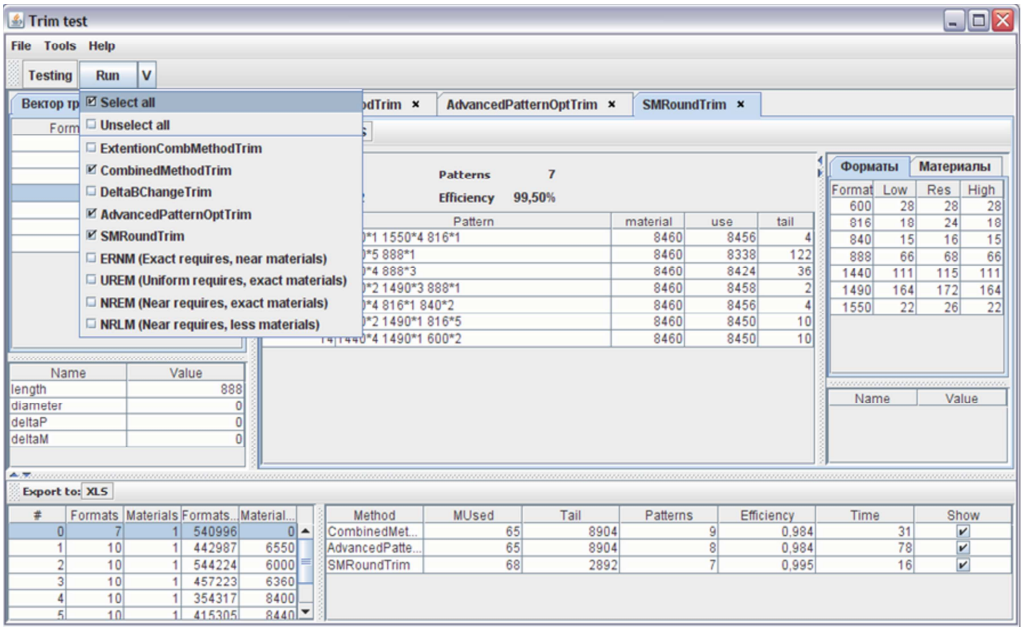


Рис. 4: Окно массового тестирования алгоритмов

3. Время выполнения — время, которое затрачивает алгоритм на свою работу.

Сравним по этим трем критериям следующие алгоритмы:

1. ERNM k=n — алгоритм выкраивает заданный вектор требований из нежестко заданного количества материалов, где k — количество использованных типов материалов.

2. СМТ — комбинированный метод решения ЗФР, использующий один тип материала, эквивалентный ERNM k=1

3. АРО — метод решения ЗФР, использующий один тип материала, который получает как можно более вырожденное решение

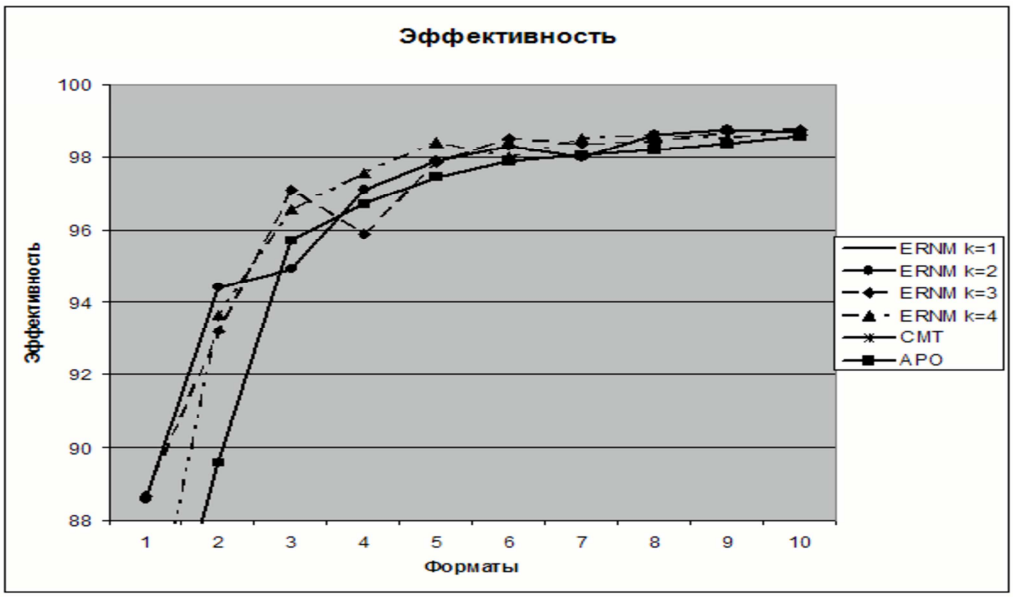


Рис. 5: Зависимость эффективности решения, от размера задачи

По графику можно заметить, что при количестве форматов от 1 до 4, ERNM k=n выдают более безотходные решения, чем СМТ и АРО. Это связано с тем, что последние для раскладки форматов используют один материал, а ERNM k=n использует k материалов, в таком случае возрастает вероятность получать наиболее безотходные раскрои, чем у СМТ и АРО. Также можно заметить, что при дальнейшем возрастании количества форматов разница между алгоритмами практически стирается, так как большее количество форматов уже проще раскладывается на одном типе материала, и преимущества от использования нескольких типов материалов пропадают.

АРО — специальный алгоритм для формирования наиболее вырожденных решений, остальные на эту задачу не нацелены. СМТ и ERNM k=n выдают решения практически равные по степени вырожденности.

В алгоритм АРО используется оптимизация по вырожденности решения, принадлежащая классу NP–алгоритмов, поэтому время, необходимое на выполнение алгоритма, растет экспоненциально и ростом количества форматов. Среди сравниваемых алгоритмов СМТ самый быстрый. Алгоритмы ERNM

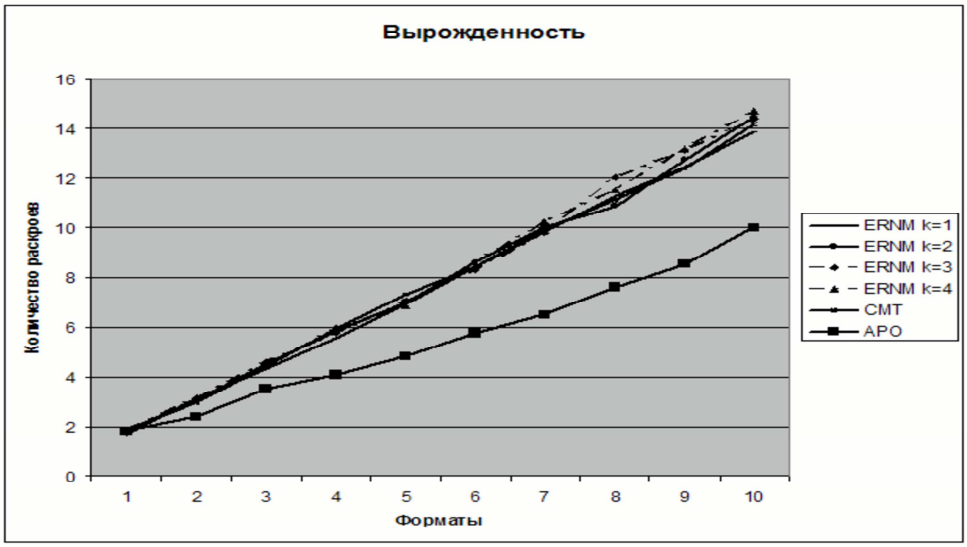


Рис. 6: Зависимость вырожденности решения, от размера задачи

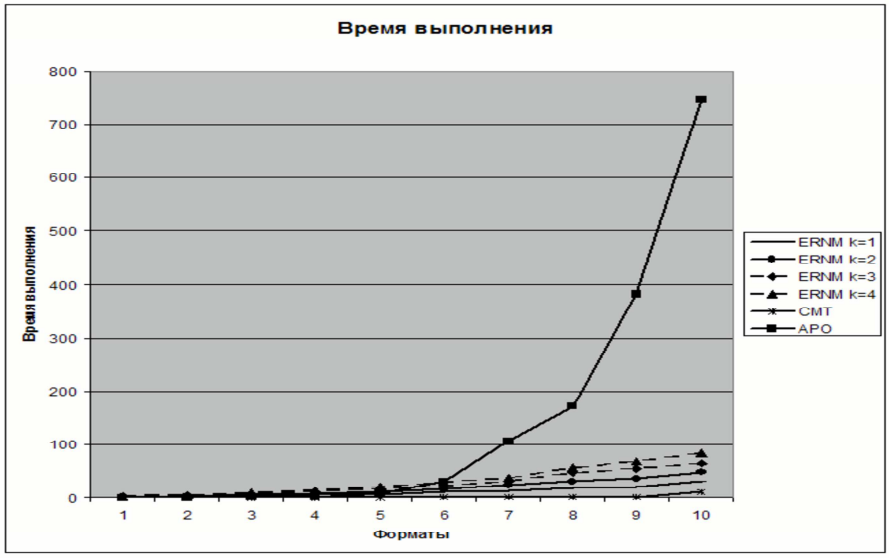


Рис. 7: Зависимость времени выполнения, от размера задачи

k=n по времени выполнения в k раз медленнее СМТ, это связано с особенностями генерации наилучших раскроев при нескольких типах материалов.

Таким образом, были разработаны общие принципы построения приложений по работе с ЗФР, которые позволили разделить разработку вычислительных алгоритмов и программного обеспечения их использующего (пользовательские интерфейсы, web–сервисы и прочее). Основываясь на этих принципах, была разработана платформа для тестирования алгоритмов решения ЗФР, эффективность которой была проверена при сравнении имеющихся алгоритмов.

Литература

1. **Никитенков В.Л., Саковнич Д.Ю.** О реализации комбинированного алгоритма решения целочисленной задачи линейного раскроя // *Вестник Сыктывкарского университета. Сер.1: Мат. Мех. Инф. 2006. Вып.6. С. 199—208.*

2. **Никитенков В.Л., Подоров А.Е.** Модификации задачи раскроя отходов // *Вестник Сыктывкарского университета. Сер.1: Мат. Мех. Инф. 2009. Вып.10. С.119-136.*

3. **Саковнич Д.Ю.** Вырожденность в задаче форматного раскроя // *Вестник Сыктывкарского ун-та, Сер.1 : Мат.Мех.Инф. 2008. Вып.8. С 75—90.*

4. **Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес** Приемы объектно-ориентированного программирования. Паттерны проектирования. СПб.: Питер, 2008. 366 с.

5. **Брюс Эккель** Философия Java. СПб.: Питер, 2009. 638 с.

Summary

Podorov A.E., Sakovnich D.Y. Platform for testing methods of solving linear cutting problem

Common ideas of building applications for solving cutting problems were developed. Using these ideas, platform for testing algorithms was built. This platform was applied for compare some algorithms.

Сыктывкарский университет *Поступила 26.03.09*