**ОГЛАВЛЕНИЕ**

[Введение 4](#_Toc58855844)

[1 Теоретические основы однокритериальной и многокритериальной оптимизации 5](#_Toc58855845)

[1.1 Основные понятия оптимизации проектных решений 5](#_Toc58855846)

[1.2 Классификации методов оптимизации 6](#_Toc58855847)

[1.3 Однокритериальная оптимизация 8](#_Toc58855848)

[1.4 Многокритериальная оптимизация 10](#_Toc58855849)

[2 Алгоритмический анализ задачи 14](#_Toc58855850)

[2.1 Постановка задачи 14](#_Toc58855851)

[2.2 Описание алгоритмов решения задач 15](#_Toc58855852)

[3 Программная реализация задачи 23](#_Toc58855853)

[3.1 Интерфейс приложения для однокритериальной оптимизации 23](#_Toc58855854)

[3.2 Интерфейс приложения для многокритериальной оптимизации 25](#_Toc58855855)

[3.3 Результат верификации приложения 28](#_Toc58855856)

[Заключение 30](#_Toc58855857)

[Список использованных источников 31](#_Toc58855858)

[Приложение А Чертёж схемы алгоритма 32](#_Toc58855859)

[Приложение Б Текст программы 33](#_Toc58855860)

# **ВВЕДЕНИЕ**

В разных ситуациях, когда нужно выбирать «одно из двух зол», прибывает практически ежедневно каждый из нас. Хорошо, если ошибка в выборе варианта не доставляет особых хлопот, но есть области человеческой деятельности, в которых ошибка может стать очень проблематичной. Из-за этих обстоятельств возникла необходимость выделить процесс принятия оптимальных решений в отдельную область науки, которая занималась бы формализацией и систематизацией этого процесса.

Процессы принятия решений лежат в основе всей целевой человеческой деятельности. Это означает, что особое внимание необходимо уделять науке принятия оптимальных решений, тем самым увеличивая процент безошибочных вариантов. Важные решения принимают опытные люди, далекие от математики и особенно от ее новых методов, и поэтому часто больше боятся проиграть, чем выиграть, используя формализацию.

Моменты принятия правильных решений, которые можно сравнить с игрой в «Русскую рулетку», уже давно прошли [1]. Сегодня для разработки такого решения требуется научный подход, поскольку потери, связанные с ошибками, слишком велики.

Оптимальные решения позволяют достичь поставленной цели с минимальными материальными и трудовыми затратами. Поэтому в настоящее время большое внимание уделяется анализу и методам принятия оптимальных решений. Чтобы процент риска при принятии решения был максимально близок к нулю, нужно подходить к исследованию со всех возможных сторон. Необходимо создать модель для одновременной оптимизации решений по нескольким критериям, что усложняет задачу оптимизации, особенно если она проводится вручную: количество альтернатив может достигать очень большого значения. Следовательно, возникает необходимость в оптимизации процесса принятия решений, что может сузить количество предлагаемых альтернатив и облегчить задачу выбора оптимальной альтернативы.

Оптимизация большого числа альтернатив по многим критериям – это достаточно трудоёмкий процесс, требующий достаточных вычислительных мощностей. Так как компьютеры тесно вошли в нашу жизнь, то логично предположить, что существуют алгоритмы, которые выполняют оптимизации различного рода автоматически на основании каких-либо экспертных оценок и, самое главное, делают это намного быстрее, чем если бы это делали люди без использования компьютеров.

# **1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ОДНОКРИТЕРИАЛЬНОЙ И МНОГОКРИТЕРИАЛЬНОЙ ОПТИМИЗАЦИИ**

## **1.1 Основные понятия оптимизации проектных решений**

Термин «оптимизация» относится к процессу выбора наилучшего варианта из всех возможных. С точки зрения инженерных расчётов методы оптимизации позволяют выбрать лучший вариант конструкции, лучшее распределение ресурсов.

Оптимизация осуществляется с помощью алгоритмов математического программирования и может быть структурной, параметрической и структурно-параметрической. В процессе оптимизации конструкции оптимизируется структура объекта, в процессе параметрической оптимизации оптимизируются параметры элементов, составляющих структуру. Эти задачи решаются с использованием алгоритмов дискретного, непрерывного и дискретно-непрерывного математического программирования соответственно.

В зависимости от количества критериев, по которым оптимизируется объект, можно различать однокритериальную и многокритериальную оптимизацию.

По наличию ограничений в целевой функции и операционных параметрах различают оптимизацию без ограничений и при наличии ограничений. Таким образом, если при синтезе усилителя необходимо, чтобы коэффициент усиления был не меньше определенного значения, то говорят о наложении ограничения на соответствующий критерий. Если в этом случае необходимо использовать названия элементов, значения которых должны находиться в определенном диапазоне, то мы имеем дело с ограничениями на рабочие параметры.

В случае параметрического синтеза с известной структурой объекта параметры элементов выбираются таким образом, чтобы минимизировать (максимизировать) целевую функцию.

Следует отметить, что существующие алгоритмы оптимизации обычно не гарантируют, что глобальный оптимум будет найден, но это не критично. Например, чтобы увеличить вероятность нахождения глобального оптимума, можно значительно увеличить количество итераций, использовать несколько алгоритмов и многократно выполнять соответствующие алгоритмы [2]. Современные передовые системы автоматизированного проектирования включают в себя модули для параметрического синтеза и оптимизации, которые, в свою очередь, очень помогают человеку в выборе решений.

Если помимо подбора параметров необходимо еще и определить структуру объекта, то мы будем уже иметь дело со структурно-параметрическим синтезом, который решается при помощи алгоритмов дискретно-непрерывного математического программирования. Если задача параметрической оптимизации сейчас решается практически для любых объектов, то развитие структурно-параметрической оптимизация сейчас находится лишь на начальной стадии развития. С теорией оптимизации тесно связаны математическое программирование, теория исследования операций, теория принятия решений, динамическое программирование. Дальнейшее развитие теории и практики оптимизации является очень важным для развития науки и техники.

Выбор оптимального решения или сравнение двух альтернативных решений осуществляется с использованием некоторой зависимой величины (функции), которая определяется проектными параметрами. Это значение называется целевой функцией или критерием качества. В процессе решения задачи оптимизации необходимо найти такие значения проектных параметров, при которых целевая функция имеет максимум (или минимум). Таким образом, целевая функция является глобальным критерием оптимальности математических моделей, с помощью которых описываются инженерные задачи.

В случае конструктивного параметра, это функция переменной, а его график представляет собой определенную кривую на плоскости. Для двоих целевая функция – это функция двух переменных, а её график – это поверхность.

Следует отметить, что целевую функцию не всегда можно представить в виде формулы. Иногда он может принимать только отдельные значения, указываться в виде таблицы и т.д. Однако в каждом случае это должна быть уникальная функция параметров проекта.

Различают основные этапы решения задачи оптимизации:

– постановка задачи. На этом этапе аналитик должен превратить слова клиента в четко сформулированную задачу;

– построить математическую модель задачи. Здесь чётко поставленная и сформулированная жизненно важная проблема формализуется математически;

– решение математической модели задачи;

– принимая решения. На этом этапе аналитик, основываясь на предыдущих шагах, должен принять лучшее решение. Принятие решений – это разновидность управленческой задачи.

## **1.2 Классификации методов оптимизации**

Классификация – это общенаучный метод систематизации знаний, направленный на упорядочение определенного набора изучаемых объектов из разных областей реальности, знаний и деятельности в систему подгрупп (классов), в которых эти объекты расположены на основе их сходства по определенным существенным свойствам.

Методы оптимизации классифицируют в соответствии с задачами оптимизации.

Локальные методы сходятся в некотором локальном экстремуме целевой функции. В случае унимодальной целевой функции этот экстремум уникален и будет глобальным максимумом или минимумом [3]. Глобальные методы имеют дело с многоэкстремальными целевыми функциями. В глобальном поиске основная задача – выявить тенденции глобального поведения целевой функции.

По критерию размерности допустимого множества, методы оптимизации делят на методы одномерной оптимизации и методы многомерной оптимизации. По виду целевой функции и допустимого множества, задачи оптимизации и методы их решения можно разделить на следующие классы:

– задачи оптимизации, в которых целевая функция и ограничения являются линейными функциями, разрешаются так называемыми методами линейного программирования;

– задачи нелинейного программирования. В свою очередь из них выделяют две частные задачи. Если выпуклые функции, то такую задачу называют задачей выпуклого программирования. В противном случае имеем дело с задачей целочисленного (дискретного) программирования.

Практически все методы минимизации функции *n* переменных основаны на многократном повторении следующих двух действий:

– выбор в области параметров некоторого направления спуска;

– спуск к минимуму вдоль выбранного направления.

Если единичный вектор выбранного направления в точке, то уравнение прямой, проходящей через эту точку в направлении, записывается в виде, где параметр *z*, соответствующий точкам на прямой (модуль *z* есть расстояние от текущей точки).

Значения функции вдоль этой прямой можно описать функцией одной переменной. Изменяя *z* производится движение вдоль этой прямой, находим точку, в которой функция имеет меньшее значение, чем в точке. Обычно производится нахождение минимума функции одной переменной.

Все многообразие методов минимизации функции *n* переменных определяется множеством способов выбора направлений и методов спуска в выбранном направлении.

Классификация методов многомерной оптимизации [4]:

– методы нулевого порядка, где при выборе направления спуска требуют вычисления только значений функции;

– методы первого порядка, где требуются вычисления (точного или приближенного) градиента функции;

– методы второго порядка, где требуются вычисления как градиента, так и матрицы вторых производных.

Также различают методы однокритериальной оптимизации (поиск оптимума единственной целевой функции) и многокритериальной оптимизации (принятия решения при многих критериях, в частности сводят векторную задачу к последовательности скалярных задач).

Рациональные методы ориентированы на некоторые математические модели оптимизируемой функции. Эвристические алгоритмы обычно не используют какую-либо модель целевой функции, а основывают процесс оптимизации на формализованной человеческой интуиции и других расплывчатых предположениях.

Методы оптимизации также подразделяются на детерминированные и стохастические. Стохастические алгоритмы используют случайные элементы при выборе направления или длины шага в процессе оптимизации. Обратите внимание, что методы стохастической оптимизации применяются к детерминированным задачам (случайность намеренно вводится в алгоритм, чтобы обеспечить достижение цели).

Стохастические алгоритмы, в свою очередь, подразделяются на (прямые) алгоритмы случайного поиска и алгоритмы аппроксимации стохастического градиента.

## **1.3 Однокритериальная оптимизация**

Цель оптимизации по одному критерию – найти лучшее решение для определенного критерия или показателя, такого как время работы, мощность и сочетание этих показателей с показателями энергопотребления или потерь мощности. Можно объединить несколько критериев в одну оптимизацию критерия, определив функцию стоимости одного критерия как взвешенную сумму нормализованных затрат, связанных с каждой метрикой.

Задачи одномерной минимизации представляют собой простейшую математическую модель оптимизации, в которой целевая функция зависит от одной переменной, а допустимым множеством является отрезок вещественной оси.

Общая постановка задачи одномерной оптимизации сводится к следующему: имеется некоторая функция *f(x)* от одной переменной *x*, необходимо найти такое значение *x\**, при котором функция *f(x)* принимает экстремальное значение. Под этим значением понимают минимальное или максимальное значения.

В общем, функция может иметь одну или несколько крайних точек. Поиск таких точек с определенной точностью обычно делится на два этапа. На первом этапе отделяются крайние точки. Это означает, что определяются участки, содержащие одну крайнюю точку, а затем уточняются до требуемой точности *ε*. Максимизация целевой функции эквивалентна минимизации противоположного значения; поэтому могут рассматриваться только задачи минимизации.

Для решения задачи минимизации функции *f(x)* на отрезке *[a, b]*, в большинстве случаев, используют приближенные методы. С их помощь можно найти решения этой задачи с необходимой точностью в результате нахождения конечного числа значений функции *f(x)* и ее производных в некоторых точках отрезка *[a, b]*. Методы, использующие только значения функции и не требующие вычисления ее производных, называются прямыми методами минимизации. Достоинством прямых методов является то, что от целевой функции не требуется дифференцируемости и, более того, она может быть не задана в аналитическом виде.

Единственное, на чем основаны алгоритмы методов прямой минимизации – это возможность определять значения *f*(*x*) в определенных точках. Среди задач математического программирования наиболее распространенными и изученными являются так называемые задачи линейного программирования (линейная оптимизация). Давайте рассмотрим этот тип задач. Они характеризуются тем, что целевая функция линейно зависит от параметров, а также тем, что ограничения, накладываемые на независимые переменные, принимают форму линейного равенства или неравенства по этим переменным. Эти задачи нередко используются на практике – например, при решении проблем, связанных с распределением ресурсов, планированием производства, организацией работы транспорта и т.д.

Общая задача линейной оптимизации заключается в нахождении максимума (минимума) линейной целевой функции. Вид целевой функции и ограничений описываются формулами (1.1-1.4).

|  |  |  |
| --- | --- | --- |
|  |  | (1.1) |

при ограничениях:

|  |  |  |
| --- | --- | --- |
|  |  | (1.2) |
|  |  | (1.3) |
|  |  | (1.4) |
|  |  |  |

где −целевая функция, критерий оптимальности или линейная форма;

*x* − вектор неизвестных;

 − коэффициенты целевой функции;

 − коэффициенты ограничений;

 − величины правых частей ограничений.

Вектор значений неизвестных *x = (x*1*, x*2*, …, xn)*, удовлетворяющих условию задачи, называется допустимым решением или допустимым планом задачи линейной оптимизации. Совокупность всех допустимых планов называется множеством допустимых планов. Допустимое решение *x\* = (x*1\**, x*2\**, …, xn\*)* называется оптимальным, если оно обеспечивает максимальное (или, в зависимости от условий задачи, − минимальное) значение целевой функции.

Решение задач линейной оптимизации может быть получено без особых трудностей (разумеется, при правильной постановке задачи). Классическим методом решения таких задач является симплексный метод. Если используются только две переменные, можно также успешно использовать графический метод решения, который имеет преимущество ясности. Очевидно, что, если *n* больше двух, графический метод не может быть применен.

При решении ряда оптимизационных задач требуется, чтобы значения неизвестных *x = (x*1*, x*2*, …, xn)* выражались в целых числах. К задачам подобного типа относятся те, в которых требуется определить необходимые для принятия решений значения физически цельных объектов (машин, агрегатов различного типа, людей, транспортных единиц и т.д. и т.п.).

Такие задачи называются задачами целочисленной оптимизации. Математическая модель задачи линейной целочисленной оптимизации также определяется формулами, но в этом случае существует дополнительное требование, чтобы все (или некоторые) неизвестные были целочисленными. Если требование целого числа применяется только к некоторым из неизвестных значений проблемы, то проблема частично называется целым числом.

## **1.4 Многокритериальная оптимизация**

Многокритериальная оптимизация или программирование – это процесс одновременной оптимизации двух или более конфликтующих целевых функций в заданной области определения [5].

Любая проблема практического решения имеет несколько желательных альтернатив, но традиционные подходы к оптимизации отдельных критериев ограничиваются одним решением. Также может быть ряд ограничений, но во всех случаях структура модели требует единого заранее определенного набора взаимных уступок. Оптимизация по нескольким критериям предлагает множество различных подходов и методов, которые можно использовать для поиска подходящих альтернатив наиболее эффективным и высококачественным образом.

Многокритериальная оптимизация используется во многих областях науки, включая инженерию, экономику и логистику, где оптимальные решения должны приниматься, когда существует компромисс между двумя целями. В таких задачах нет общего понятия оптимальности, поскольку существуют разные объективные критерии, которые могут противоречить друг другу. На практике часто не существует единого критерия оптимальности для измерения качества решения. Скорее, последнее определяется ценностью решения по каждому объективному критерию. Фактически, практикующий специалист ищет решение, имеющее приемлемую ценность, и на практике могут быть разные допуски к качеству решения, предусмотренного для разных целевых функций: для некоторых объективных критериев решения, далекие от оптимальных, могут быть приемлемыми. Традиционный подход оптимальности направлен на создание всех возможных решений без доминирования над всеми критериями оптимальности. Часто это занимает неприемлемое количество времени. Кроме того, не очевидно, как выбрать подходящее решение из оптимального набора возможных решений, который может быть очень большим. Здесь мы предлагаем новый подход и называем его настройкой многопороговой оптимизации, которая учитывает различные требования для разных объективных критериев и, следовательно, является более гибкой и часто может быть решена более эффективно.

Задача многокритериальной оптимизации – это задача оптимизации, которая включает в себя несколько целевых функций [6]. С математической точки зрения задача многокритериальной оптимизации может быть сформулирована следующей функцией (формула 1.5):

|  |  |  |
| --- | --- | --- |
|  |  | (1.5) |

Относительная важность целей обычно неизвестна до тех пор, пока не будут определены все основные свойства системы и полностью не поняты все возможные взаимосвязи.

По мере увеличения числа возможных целей становится ясно, что эти отношения образуют сложную структуру, и их становится труднее идентифицировать. В этом случае многое зависит от интуиции исследователя и его способности точно выразить определенные предпочтения в процессе оптимизации.

Таким образом, стратегия построения многокритериальной оптимизации состоит, прежде всего, в умении адекватно определить постановку задачи так, чтобы эта задача допускала свое решение, а также выразить необходимые предпочтения в виде числовых зависимостей, при сохранении реальности поставленной задачи.

Задача многокритериальной оптимизации формулируется следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | (1.6) |
|  |  |  |

где  – это *n* целевых функций.

Векторы решений приведены в формуле (1.7) и относятся к не пустой области определения *S*:

|  |  |  |
| --- | --- | --- |
|  |  | (1.7) |

Задача многокритериальной оптимизации – найти вектор целевых переменных, удовлетворяющий наложенным ограничениям, и оптимизировать вектор-функцию, элементы которой соответствуют целевым функциям. Эти функции образуют математическое описание удовлетворительного критерия и обычно противоречат друг другу. Таким образом, «оптимизация» означает поиск решения, при котором значение целевых функций для проблемы было бы приемлемым.

Итальянский экономист В. Парето сформулировал один из самых распространенных экономических критериев оптимальности. Он формулируется очень просто: «Следует считать, что любое изменение, которое никому не причиняет убытков и которое приносит некоторым пользу, является улучшением».

Критерий Парето имеет очень широкий экономический смысл и очень часто используется для решения сложных экономических задач. Его можно использовать, когда оптимизация одного набора показателей, характеризующих объект, не должна ухудшать другой набор не менее важных показателей. Критерий Парето не применяется к ситуациям, когда экономический эффект одних связан с потерями других. Область оптимальных решений Парето называется областью компромисса.

Алгоритм выделения области Парето проводится в несколько следующих шагов:

1) выбирается проект Пi, полагая i = 1;

2) проект Пi сравнивается с остальными по всем показателям и отмечаются те из них, которые строго хуже, чем Пi;

3) отмеченные проекты не могут принадлежать области Парето и из дальнейшего рассмотрения исключаются;

4) полагается i = i + 1;

5) если  и проект Пi уже был отмечен на предыдущих итерациях, то выполняется переход к четвёртому шагу;

6) если  и проект еще не помечен, то производится переход ко второму шагу;

7) если , где n – число сравниваемых проектов, то осуществляется переход к восьмому шагу;

8) оставшиеся не отмеченными проекты образуют множество эффективных решений (область Парето).

Если использовать дополнительную информацию, например, об относительной важности показателей, то можно получить дальнейшее уточнение мест проектов.

Часто, когда необходимо сделать выбор из альтернативы, возникает необходимость в измерениях с использованием специальных шкал. Такие шкалы используются в случаях, когда оценки субъективны. Эти шкалы называются психофизическими и позволяют формализовать систему предпочтений эксперта. Психофизические шкалы задаются функциями особого вида - функциями желательности и устанавливают соответствие между естественными значениями показателей в физических шкалах и психологическими параметрами - субъективными оценками «ценности» этих значений. К физическим показателям объекта могут относиться экономические, технические, технологические, экологические, социальные, эстетические и другие параметры.

Самым известным и широко используемым является метод Харрингтона. Введение шкалы желательности позволяет свести исходную многокритериальную задачу решения с критериями разных измерений к многокритериальной задаче с критериями, измеряемыми по той же шкале. Возможность широкого применения этого метода в задачах оценки и оптимизации подтверждается исследованиями ученых в различных областях.

Многообразие областей применения метода указывает на возможность его применения при оценке уровня техносферной безопасности на территории. Метод Харрингтона для описания частных критериев и ограничений при решении многокритериальных задач позволяет установить соответствие между лингвистическими оценками удобства значений оценочного показателя и числовыми диапазонами предпочтений. Интервалы предполагают значения, которые непрерывно увеличиваются от нуля до единицы, так как соответствующий параметр качества изменяется от наименьшего до наиболее желательного. Конкретный вид функций желательности задается лицом, принимающим решения, исходя из его субъективных представлений. Затем путем свертки частных функций желательности строится глобальный критерий качества процесса, максимизация которого составляет оптимум.

Метод Харрингтона обладает следующими преимуществами:

– является количественным;

– выражается одним числом, т.е. является единой функцией;

– является однозначным, т.е. заданному набору значений частных параметров оценки соответствует одно значение функции;

– универсальный и может использоваться в различных областях;

– всесторонне характеризует объект, т.е. соответствует требованию полноты;

– обеспечивает простой путь преобразования показателей с помощью одного графика для всех критериев;

– «нейтральна» при обобщении, на конечный результат при построении шкалы желательности влияют только личные предпочтения;

– является адекватным методом. Адекватность частичных и обобщенных функций желательности следует понимать как их эквивалентность измеренным значениям параметров оптимизации в том смысле, что с ними могут выполняться все вычислительные действия, определенные в наборе значений параметров оптимизации.

Процесс оценки состоит из следующих этапов:

– определение перечня частных параметров оценки, т.е. критериев, по которым будет оцениваться объект (количественных, качественных);

– установление границы допустимых значений. Это удобнее, когда есть четкие ограничения. Однако из-за нового характера исследования или отсутствия данных допустимые значения для отдельных параметров могут быть недоступны. Тогда пределы для таких показателей устанавливает эксперт, исходя из опыта и интуиции экспериментатора, то есть оценка субъективна. Устанавливая предел допустимых значений, следует учитывать, что ограничения могут быть установлены как с одной, так и с двух сторон. Односторонние ограничения устанавливаются, если улучшение функции желательности происходит только при однонаправленном изменении показателя (уменьшении или увеличении).

– выбор шкалы желательности. Для получения шкалы желательности удобно использовать таблицу соответствий, разработанную между отношениями предпочтения в эмпирической и числовой (психологической) системах. Масштаб должен быть одинаковым для всех объединенных параметров (ответов). Однако стандартные значения шкалы желательности не являются обязательными;

– определение уровня желательности в соответствии со шкалой Харрингтона для каждого параметра оценки.

# **2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ЗАДАЧИ**

## **2.1 Постановка задачи**

Для решения поставленной задачи требуется разработать приложение для механического цеха, которое планирует установить на входных воротах тепловую завесу в количестве не менее 3. Для начала установки нужно определить наиболее оптимальные варианты для приобретения. При этом присутствуют следующие ограничения:

– годовое время работы завес (не более 5000 часов);

– общая стоимость завес (не более 200 миллионов рублей);

– суммарная электрическая мощность (не более 120 кВт).

Для решения поставленной задачи необходимо разработать приложение, которое выполнит однокритериальную и многокритериальную оптимизацию для выбора наилучшего варианта установки завес.

Приложение должно иметь следующие функции:

– производить однокритериальную оптимизацию по удельному расходу электроэнергии;

– производить проверку решения методом полного перебора;

– производить многокритериальную оптимизацию.

Задача многокритериальной оптимизация делится на следующие подзадачи:

– перевод качественных оценок в числовую форму по шкале Харрингтона;

– определение значимости (веса) критериев методом экспертного анализа (метод Ранга) для двух экспертов;

– отбор бесперспективных альтернатив с помощью множества Парето;

– определение лучших альтернатива методом анализа иерархий.

Исходными данными является информация с характеристиками производственных линий по производству зефира, которая должна быть найдена в интернете (таблица 2.1).

Таблица 2.1 – Информация о производственных линиях зефира

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Название** | **Стоимость (тыс. руб.)** | **Расход электроэнергии (кВт\*ч/т)** | **Расход питьевой воды (м^3)** | **Необходимость водяного пара** | **Месячное время работы линии (ч)** | **Выпуск (количество) (т в год)** |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Rostehno LPM 80 | 80 | 130 | 200 | Есть | 400 | 1870 |

Продолжение таблицы 2.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Rostehno LPM 80 | 80 | 130 | 200 | Есть | 400 | 1870 |
| Rostehno LPM 100 | 130 | 230 | 300 | Есть | 200 | 3100 |
| ШЗД-1 | 78 | 144 | 120 | Есть | 400 | 1950 |
| Таурас-Феникс SR 500 | 50 | 450 | 60 | Есть | 250 | 2600 |
| Тандер АП 3012 | 60 | 195 | 255 | Есть | 275 | 1990 |
| neofood | 60 | 190 | 50 | Нет | 210 | 2000 |

Для оценки эффективности работы приложения, необходимо увеличить число альтернатив.

## **2.2 Описание алгоритмов решения задач**

***2.2.1*** Задача однокритериальной оптимизации решается методом полного перебора. Полный перебор – это один из наиболее простых методов оптимизации, который универсален и подходит не только для задачи оптимизации, но и для многих других. Суть данного метода заключается в последовательном переборе всех элементов, имеющих различные значения некоторых характеристик. В ходе перебора находится элемент, у которого значение параметра лучше всего подходит под изначальные требования к результату. Недостаток данного метода – это долгое время выполнения при больших количествах обрабатываемых элементов, поэтому данный способ применяется в первую очередь для однокритериальной оптимизации, где параметром выступает единственный критерий. Основная сложность при реализации метода перебора – это реализация максимально эффективного алгоритма для наиболее быстрого решения.

Перед использованием метода полного перебора в задаче однокритериальной оптимизации нужно сначала сделать фильтрацию альтернатив по заданным параметрам:

– годовое время работы не должно превышать определённое значение (в данном случае 5000 часов);

– общая стоимость не должна превышать определённое значение (в данном случае 200 миллионов рублей);

– суммарная электрическая мощность не должна превышать определённое значение (в данном случае 120 киловатт).

После фильтрации альтернатив наступает этап нахождения лучшей альтернативы по какому-то заданному критерию, в данном случае им является удельный расход электроэнергии. Для этого нужно список альтернатив отсортировать по этому критерию и получится список, в котором в начале находятся альтернативы с наиболее низким удельным расходом электроэнергии, а в конце оставшиеся альтернативы с больших расходом электроэнергии.

***2.2.2*** Для решения задачи многокритериальной оптимизации нужно выполнить несколько этапов, подзадач.

Как и в случае с однокритериальной оптимизацией сначала требуется провести фильтрацию альтернатив по заданным параметрам, описанным выше, для получения альтернатив, которые будут удовлетворять заданным к ним требованиям.

Принятие решений как научное направление характеризуется тем, что его практически невозможно полностью формализовать, так как в процессе принятия решений используются процедуры многокритериального анализа альтернатив в условиях неполной информации о внешних факторах, влияющих на эффективность решений, учитывается субъективное мнение экспертов. Основные проблемы, возникающие при многокритериальной оптимизации:

– противоречивость критериев. Как правило, улучшение по одному критерию приводит к ухудшению по другому критерию. Например, более производительный компьютер стоит дороже;

– при многокритериальной оптимизации используются разнородные оценки: числовые, качественные(«быстрый процессор», «отличная машина» и т.п.), оценки вида «да-нет», балльные оценки, оценки в виде ранжирования и т.д., которые к тому же отличаются по размерности (например, цена – в рублях, производительность компьютера – в миллиардах операций в секунду, объём и т.п.);

– критерии могут сильно различаться по важности;

– крайне сложно (иногда и невозможно) определить аналитическую зависимость между критериями.

Один из вариантов перевода качественных оценок в числовую форму – это перевод по шкале Харрингтона (таблица 2.2).

Таблица 2.2 − Шкала Харрингтона

|  |  |
| --- | --- |
| **Качественная оценка** | **Числовая оценка** |
| Очень плохо | 0,0 – 0,2 |
| Плохо | 0,2 – 0,36 |
| Удовлетворительно | 0,36 – 0,63 |
| Хорошо | 0,63 – 0,8 |
| Отлично | 0,8 – 1,0 |

В случае, если две альтернативы имеют одинаковую оценку «хорошо», но по мнению эксперта или ЛПР вторая альтернатива лучше, то первой альтернативе можно назначить оценку 0,7, а второй – 0,8.

Для оценок, имеющих вид «да-нет», используют следующую шкалу: «да» – 0,67; «нет» – 0,33.

Оценки различного рода, представленные различными системами измерений, заменяются экспертными оценками, представленными в виде балльных оценок, в долях единицы, в виде парных сравнений, в виде ранжирования и т.д.

Нормирование критериев. Числовые оценки из произвольного диапазона приводят к единому масштабу, то есть оценкам, лежащим в диапазоне [0;1]. При этом, как правило, лучшей оценке соответствует большее значение. Формула нормализации:

|  |  |  |
| --- | --- | --- |
|  |  | (2.1) |

где *x* – список всех значений;

*z* – нормализованные значения, лежащие в диапазоне [0;1];

*i* – номер значения в списке.

При решении сложных и сложных задач, особенно в условиях неопределенности и неполноты информации, широко используются методы экспертного анализа. Идея экспертизы заключается в том, что к получению оценок привлекаются люди, разбирающиеся в этой области, эксперты, которые проводят логико-интуитивный анализ любого вопроса, чтобы вынести решение. Экспертные заключения обрабатываются определенным образом с использованием специальных математических процедур. В результате получаются так называемые экспертные оценки. Следует отметить, что экспертная оценка не является решением. Это только информация, которая поможет ЛПР найти оптимальное решение. В целом предпочтения лиц, принимающих решения, могут не совпадать с предпочтениями экспертов. Однако суждения экспертов и их советы помогают ЛПР, критически понимать различные точки зрения, уточнить или изменить свою систему предпочтений и тем самым снизить вероятность принятия неоптимальных решений.

Используя эти методы, для каждой пары альтернатив определяется рейтинг превосходства одной альтернативы над другой; Этот рейтинг может быть выставлен непосредственно человеком или рассчитан на основе оценок альтернатив по отдельным критериям. На основе этих сравнений определяется лучшая альтернатива. Метод попарного сравнения основан на попарном сравнении альтернатив. Для каждой пары альтернатив эксперт указывает, какая из альтернатив предпочтительнее (лучше, важнее и т.д.). Существует ряд алгоритмов, реализующих метод парных сравнений: они различаются по количеству используемых экспертных оценок (индивидуальные и коллективные оценки), по шкалам сравнения альтернатив и т.д.

Метод Саати основан на сравнении альтернатив, выполняемом экспертом. Для каждой пары альтернатив эксперт указывает, в какой степени одна из них предпочтительнее другой.

Решение на основе метода Саати принимается в следующем порядке.

Экспертом заполняется матрица парных сравнений размером *N*x*N*, где *N* – количество альтернатив. Матрица заполняется по правилам, приведённым в таблице 2.3.

Таблица 2.3 − Правила заполнения матрица по методу Саати

|  |  |
| --- | --- |
| ***Xij*** | **Значение** |
| 1 | *i*-я и *j*-я альтернативы примерно равноценны |
| 3 | *i*-я альтернативы немного предпочтительнее *j*-й |
| 5 | *i*-я альтернативы предпочтительнее *j*-й |
| 7 | *i*-я альтернативы значительно предпочтительнее *j*-й |
| 9 | *i*-я альтернативы явно предпочтительнее *j*-й |

Если *i*-я альтернатива менее предпочтительна, чем *j*-я, то указываются обратные оценки (1/3, 1/5, 1/7, 1/9). Могут использоваться промежуточные оценки (2, 4, 6, 8 и 1/2, 1/4, 1/6, 1/8); например, если *i*-я альтернатива совсем немного лучше *j*-й, то можно использовать оценку *Xij*=2 (тогда *Xji*=1/2). На главной диагонали ставятся единицы. Пример заполненной матрицы парных сравнений приведён в таблице 2.4.

Таблица 2.4 − Пример матрицы парных сравнений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Альтернатива** | ***A*1** | ***A*2** | ***A*3** | ***A*4** | ***A*5** |
| *A*1 | 1 | 7 | 3 | 9 | 1 |
| *A*2 | 1/7 | 1 | 1/5 | 3 | 5 |
| *A*3 | 1/3 | 5 | 1 | 5 | 1/3 |
| *A*4 | 1/9 | 1/3 | 1/5 | 1 | 1/7 |
| *A*5 | 1 | 1/5 | 3 | 7 | 1 |

Далее находятся цены альтернатив – средние геометрические строк матрицы, это означает, что элементы строки перемножаются, и из их произведения извлекается корень *N*-й степени (также для упрощения расчётов в качестве цен альтернатив можно использовать суммы строк матрицы сравнений):

|  |  |  |
| --- | --- | --- |
|  |  | (2.2) |

где .

Следующий шаг подразумевает нахождение сумм цен альтернатив по формуле 2.3.

|  |  |  |
| --- | --- | --- |
|  |  | (2.3) |

Последним шаг метода попарных сравнений является нахождение весов альтернатив. Формула для вычисления весов альтернатив представлена далее:

|  |  |  |
| --- | --- | --- |
|  |  | (2.4) |

где .

Наиболее предпочтительной, по мнению эксперта, является альтернатива, имеющая максимальный вес. Если независимых экспертов несколько, можно рассчитать итоговый вес альтернативы множеством способов, учитывающих особенности экспертов, их знаний, либо сделать алгоритм проще и находить среднее значение веса альтернативы на основе весов всех экспертов.

Проверка экспертных оценок на непротиворечивость. Проверка позволяет выявить ошибки, которые мог допустить эксперт при заполнении матрицы парных сравнений. Ошибки (противоречия) могут быть следующими: например, эксперт указывает, что альтернатива *A*1 хуже, чем *A*2, а альтернатива *A*2 хуже, чем *A*3; но при этом эксперт указывает также, что *A*1 лучше, чем *A*3. Пример матрицы парных сравнений, содержащей такую ошибку, приведен в таблице 2.5.

Таблица 2.5 − Пример ошибочной матрицы парных сравнений

|  |  |  |  |
| --- | --- | --- | --- |
| **Альтернатива** | ***A*1** | ***A*2** | ***A*3** |
| *A*1 | 1 | 1/3 | 2 |
| *A*2 | 3 | 1 | 1/5 |
| *A*3 | 1/2 | 5 | 1 |

Возможны также ошибки следующего вида: эксперт указывает, что альтернатива *A*1 значительно хуже, чем *A*2, а альтернатива *A*2 значительно хуже, чем *A*3, но при этом эксперт указывает также, что *A*1 лишь немного хуже, чем *A*3. Пример матрицы парных сравнений с такой ошибкой приведен в таблице 2.6.

Таблица 2.6 − Пример ошибочной матрицы парных сравнений

|  |  |  |  |
| --- | --- | --- | --- |
| **Альтернатива** | ***A*1** | ***A*2** | ***A*3** |
| *A*1 | 1 | 1/7 | 1/3 |
| *A*2 | 7 | 1 | 1/6 |
| *A*3 | 3 | 6 | 1 |

Для проверки на непротиворечивость сначала нужно найти суммы столбцов матрицы парных сравнений по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (2.5) |

где .

Далее рассчитывается вспомогательная величина *λ* путем суммирования произведений сумм столбцов матрицы на веса альтернатив:

|  |  |  |
| --- | --- | --- |
|  |  | (2.6) |

Затем находится величина, которая называется индексом согласованности:

|  |  |  |
| --- | --- | --- |
|  |  | (2.7) |

В зависимости от размерности матрицы парных сравнений находится величина случайной согласованности (значения приведены в таблице 2.7).

Таблица 2.7 − Величины случайной согласованности

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***N*** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|  | 0,58 | 0,9 | 1,12 | 1,24 | 1,32 | 1,41 | 1,45 | 1,49 |

После нахождения индекса согласованности рассчитывается отношение согласованности:

|  |  |  |
| --- | --- | --- |
|  |  | (2.8) |

Если отношение согласованности превышает 0,2, то требуется уточнение матрицы парных сравнений, иначе матрица будет содержать оценки, которые противоречат друг другу.

Следующее действие в алгоритме многокритериальной оптимизации – это выбор множества Парето-оптимальных решений, который представляет собой отбор перспективных альтернатив, из которых затем отбираются лучшие альтернативы.

Выбор множества Парето производится следующим образом. Все альтернативы попарно сравниваются друг с другом по всем критериям. Если при сравнении каких-либо альтернатив (обозначим их как *Yi* и *Yj*) оказывается, что одна из них не лучше другой ни по одному критерию, то её можно исключить из рассмотрения.

Представим рассмотренную процедуру отбора в виде алгоритма. Пусть множество возможных альтернатив *Y* состоит из конечного числа альтернатив и имеет вид: . Необходимо сформировать новое множество альтернатив *P*(*Y*) путём исключения альтернатив, которые не превосходят ни по одному критерию остальные альтернативы.

На первом шаге положим . То есть текущее множество Парето-оптимальных альтернатив совпадает с исходным множеством.

На втором шаге нужно проверить неравенства  по всем критериям. Если оно истинно, то нужно удалить из *P*(*Y*) альтернативу  и перейти к четвёртому шагу. В противном случае перейти к третьему шагу.

На третьем шаге проверяются неравенства  по всем критериям. Если оно истинно, то из *P*(*Y*) удаляется из .

На четвёртом шаге если , то  и нужно вернуться на второй шаг. В противном случае нужно перейти к пятому шагу.

На последнем шаге алгоритма, если , то ,  и затем вернуться к второму шагу. В противном случае нужно завершить вычисления, так как множество Парето-оптимальных альтернатив сформировано.

Как правило, во множество Парето входит несколько альтернатив. Поэтому выбор множества Парето не обеспечивает принятия окончательного решения (выбора одной лучшей альтернативы), однако позволяет сократить количество рассматриваемых альтернатив, т.е. упрощает принятие решения.

Последним алгоритмом в решении задачи многокритериальной оптимизации является метод ЭЛЕКТРА. Метод ЭЛЕКТРА (*ELECTRE* – *Elimination* *Et* *Choice* *Traduisantla* *Realite* – исключение и выбор, отражающие реальность) предложена профессором Б. Руа (Франция). Метод предназначен для решения задач, в которых из имеющегося множества альтернатив требуется выбрать заданное количество лучших альтернатив с учетом их оценок по нескольким критериям, а также важности этих критериев. В зависимости от того, каким образом вычисляется важность критериев, метод ЭЛЕКТРА имеет три модификации –ЭЛЕКТРА I, ЭЛЕКТРА II, ЭЛЕКТРА III.

Основная идея метода заключается в следующем. Для каждой пары альтернатив (*Xi* и *Xj*) выдвигается предположение (гипотеза) о том, что альтернатива *Xi* лучше, чем *Xj*. Затем для каждой пары альтернатив находятся два индекса: индекс согласия (величина, подтверждающая предположение о превосходстве *Xi* над *Xj*) и индекс несогласия (величина, опровергающая это предположение). На основе анализа этих индексов выбирается одна или несколько лучших альтернатив («ядро» альтернатив).

Алгоритм метода состоит из следующих шагов (ЭЛЕКТРА I).

На первом шаге выполняется переход к безразмерному виду критериев.

На втором шаге определяются индексы согласия:

|  |  |  |
| --- | --- | --- |
|  |  | (2.9) |

где ;

;

 – вес *i*-го критерия, ;

*N* – количество альтернатив;

*I+*– подмножество критериев, по которым *j*-я альтернативы хуже *k-*й.

На третьем шаге определяются индексы несогласия:

|  |  |  |
| --- | --- | --- |
|  |  | (2.10) |

где *pik*, *pij* – безразмерные оценки;

*I–*– подмножество критериев, по которым *j*-я не превосходит *k-*ю.

На четвёртом шаге алгоритма находится предельное значение индекса согласия:

|  |  |  |
| --- | --- | --- |
|  |  | (2.11) |

На пятом шаге алгоритма находится предельное значение индекса несогласия:

|  |  |  |
| --- | --- | --- |
|  |  | (2.12) |

На последнем шестом шаге выделяют лучшие альтернативы или «ядро альтернатив» (формулы 2.13 и 2.14 соответственно).

|  |  |  |
| --- | --- | --- |
|  |  | (2.13) |

где  – пороговое значение индекса согласия (как правило значение 0,5).

|  |  |  |
| --- | --- | --- |
|  |  | (2.14) |

где  – пороговое значение индекса несогласия (как правило значение 0,5).

В методе ЭЛЕКТРА II индексы согласия подсчитываются тем же способом, что и в методе ЭЛЕКТРА I. Отличием является то, что в нем задаются два уровня для индекса согласия  и два уровня индекса несогласия , а так же вводятся два отношения предпочтения  и  между альтернативами. В методе ЭЛЕКТРА III используются псевдокритерии и числовые бинарные отношения.

Полученные альтернативы в результате работы метода ЭЛЕКТРА являются решением задачи многокритериальной оптимизации.

Чертёж схемы алгоритма многокритериальной оптимизации приведён в приложении А.

# **3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ЗАДАЧИ**

## **3.1 Интерфейс приложения для однокритериальной оптимизации**

Разработанное приложение является программой с графическим пользовательским интерфейсом. Программа написана на языке программирования *Kotlin* (версия языка для *Java* *Virtual* *Machine*) с библиотекой для создания графического интерфейса *TornadoFX* (надстройка над библиотекой *JavaFX* для *Kotlin*). Язык *Kotlin* выбран, потому что с его помощью можно создавать кроссплатформенные приложения без изменений под конкретную платформу или операционную систему. Приложение разработано с использованием объектно-ориентированной парадигмы. Разработанное приложение позволяет проводить однокритериальную и многокритериальную оптимизации. Текст программы представлен в приложении Б.

При запуске приложения пользователь видит окно программы, показанное на рисунке 3.1. Графический интерфейс приложения построен на основе сетки, благодаря чему интерфейс адаптивно изменяется под различные формы окна и также при различных разрешениях дисплеев, что добавляет комфорта при использовании программы на разных компьютерах и при разных разрешениях мониторов.

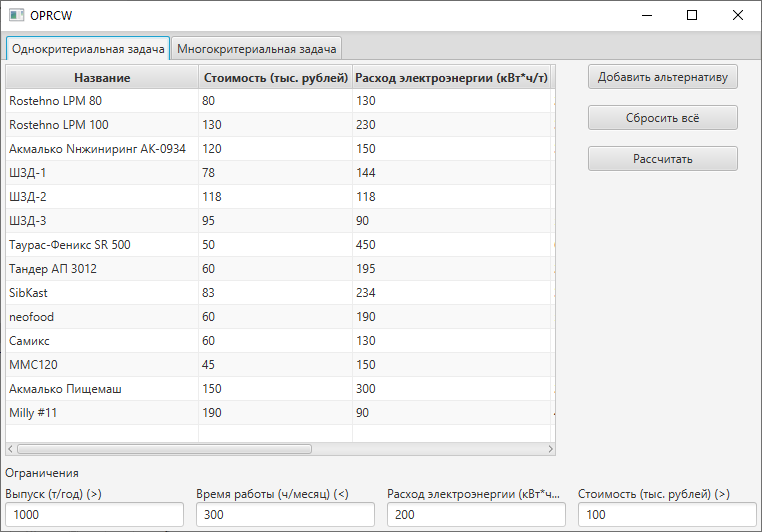


Рисунок 3.1 – Окно программы сразу после запуска

Изначально пользователь находится во вкладке задачи однокритериальной оптимизации. В ней содержится список альтернатив (линий по производству зефира) в табличном виде. Каждая строка таблицы содержит несколько столбцов, содержащих параметры конкретной производственной линии:

– столбец с названием линии;

– столбец со значением стоимости линии;

– столбец со значением расхода электроэнергии линией;

– столбец со значением расхода воды линией;

– столбец с информацией о необходимости использования водяного пара;

– столбец с временем работы линии в месяц;

– столбец с информацией о выпуске продукции в год.

Таблица со всеми столбцами и некоторыми альтернативами (линия по производству зефира) показана на рисунке 3.2.

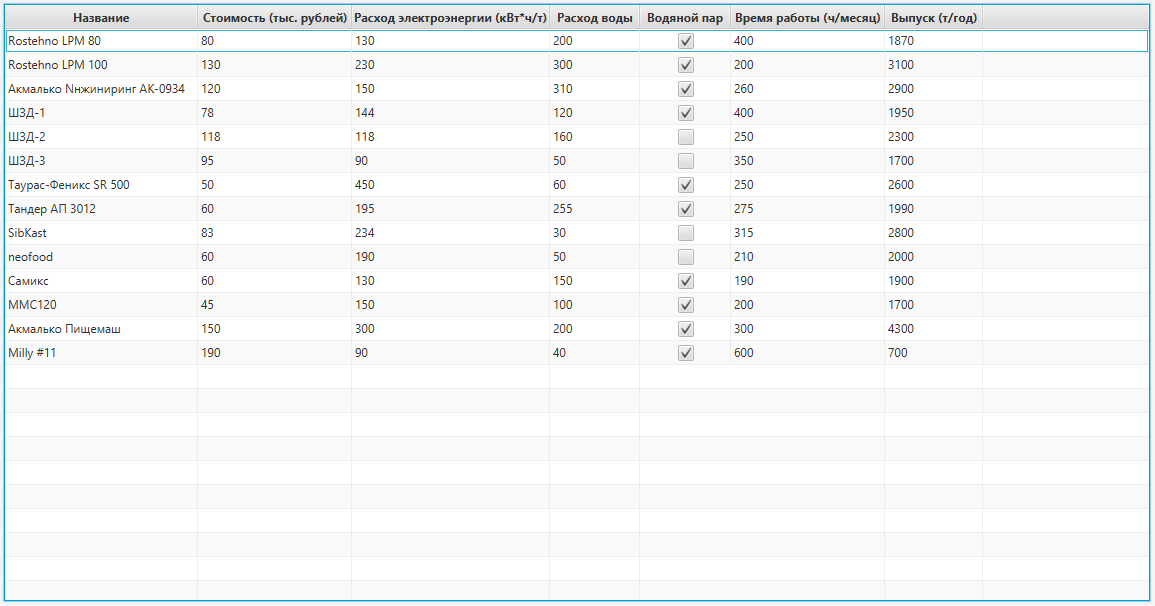


Рисунок 3.2 – Список альтернатив в табличном представлении

Для быстрого добавления, изменения и удаления альтернатив реализованы соответствующие функции. Для добавления альтернативы нужно нажать на кнопку «Добавить альтернативу» в правом верхнем углу. Для изменения параметра нужно два раза нажать левой кнопкой мыши на нужный параметр в интересующей строке, далее ввести корректное значение, после чего для сохранения нового значения нажать клавишу *Enter*. Для удаления альтернативы, нужно найти её в таблице, нажать в той строке в любом месте правой кнопкой мыши, после чего появится выпадающий список, в котором требуется нажать на пункт с текстом «Удалить».

Также для задачи однокритериальной оптимизации по условию есть ограничения для некоторых параметров. Данные ограничения можно изменять в соответствующих полях ввода внизу окна программы. По условию заданы ограничения на выпуск (количество) производимой продукции, на время работы, на расход электроэнергии и на стоимость производственной линии. Для каждого из этих параметров есть отдельные поле ввода, в которых по умолчанию заданы значения, которые даны в задании (поля ввода ограничений показаны на рисунке 3.3).

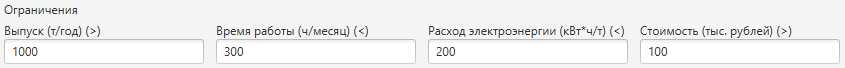


Рисунок 3.3 – Поля ввода ограничений некоторых параметров

производственных линий по изготовлению зефира

Для сброса всех параметров к значениям по умолчанию нужно нажать на кнопку «Сбросить всё». После нажатия список альтернатив будет заново считан из файла, все добавленные альтернативы не будут сохранены. Ограничения также сбросятся к стандартным значениям.

Для решения задачи однокритериальной оптимизации надо нажать на кнопку «Рассчитать». Результат расчёта показан на рисунке 3.4.

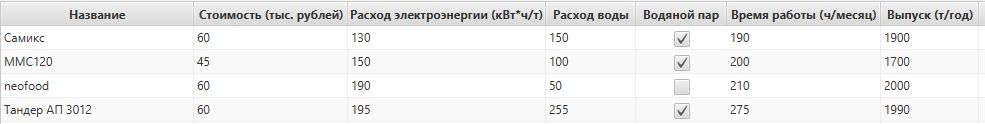


Рисунок 3.4 – Результат нахождения лучших альтернатив в задаче

однокритериальной оптимизации по критерию расхода электроэнергии

В результате расчёта производится фильтрация по установленным в момент нажатия значениям ограничений параметров, далее с помощью метода полного перебора из оставшихся альтернатив будут найдены лучшие альтернативы по критерию расхода электроэнергии и показан результат в таблице, где альтернативы будут отсортированы от лучше к худшей, сверху вниз в списке.

## **3.2 Интерфейс приложения для многокритериальной оптимизации**

Во второй вкладке с названием «Многокритериальная задача» находится интерфейс для решений задачи многокритериальной оптимизации. Исходные данные для этой задачи идентичны данным задачи однокритериальной оптимизации. Окно программы в этой вкладке показана на рисунке 3.5. Также как и в первой вкладке, в этой вкладке присутствует таблица со списком альтернатив, которые при запуске считываются из файла, таблица имеет тот же функционал по добавлению, изменению и удалению альтернатив. Есть кнопка «Добавить альтернативу» для добавления новой альтернативы в список. Также есть кнопка для сброса значений к стандартным с таким же названием – «Сбросить всё». Присутствуют поля ввода для применения ограничений к альтернативам при расчёте.

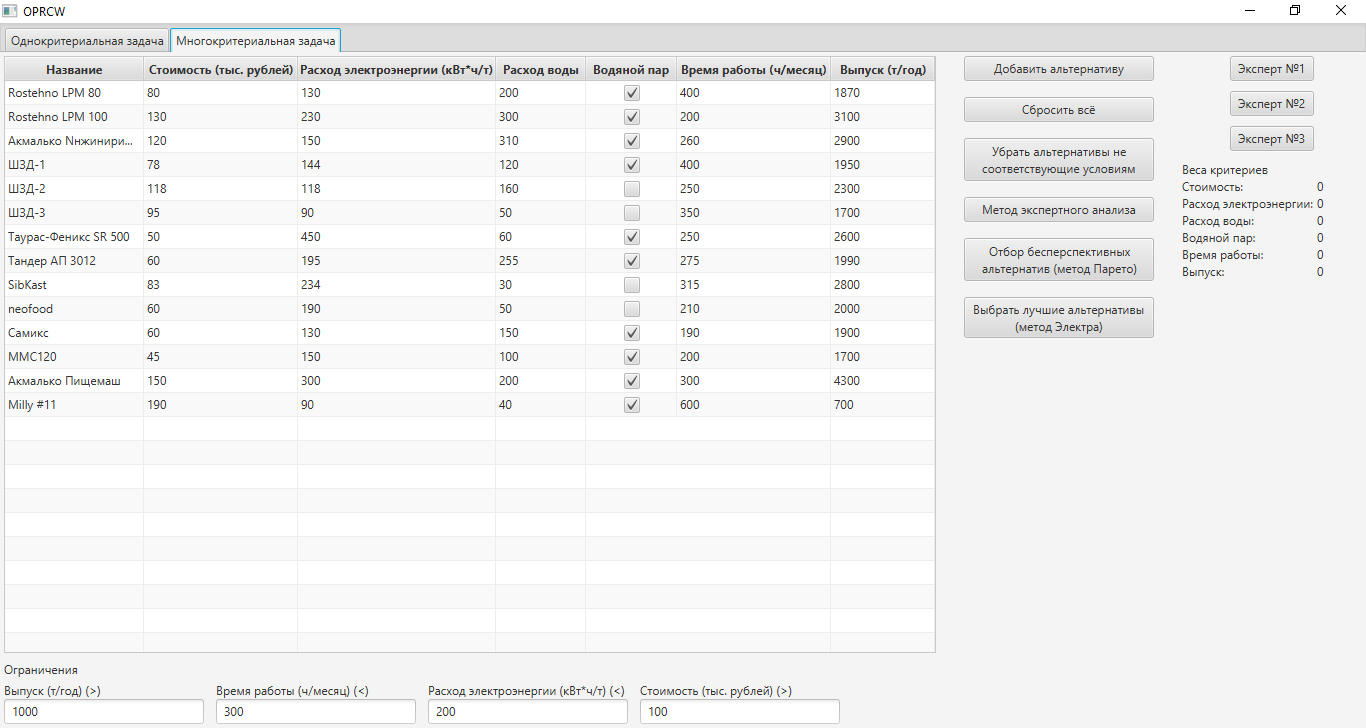


Рисунок 3.5 – Окно программы во вкладке «Многокритериальная задача» со стандартными значениями

Для каждого этапа решения задачи многокритериальной оптимизации создана отдельная кнопка. Изначально, как и в задаче однокритериальной оптимизации, из списка альтернатив требуется удалить альтернативы, не подходящие под ограничения, для этого нужно нажать на кнопку «Убрать альтернативы не соответствующие условиям», после чего программа обновит список и в нём останутся только альтернативы, подходящие под заданные условия. Результат показан на рисунке 3.6.



Рисунок 3.6 – Список альтернатив после удаления из него тех, которые не

подходят под ограничения

Следующим этапом решения будет нахождение альтернатив, которые удовлетворяют условиям метода Парето. Для запуска этого метода нужно нажать на кнопку «Отбор бесперспективных альтернатив (метод Парето)», после чего, как и в случае с удалением альтернатив по заданным ограничениям, обновится список и в нём останутся только подходящие под все условия альтернативы.

Следующий этап – это нахождение весов всех критериев. Для этого в программе можно ввести оценки для критериев в матрицу парных сравнений для каждого из трёх экспертов, после чего рассчитать итоговые оценки на основании оценок экспертов. Изначально оценку каждого эксперта заданы, то есть не обязательно их изменять. Чтобы изменить оценки критериев, нужно нажать на одну из трёх кнопок, каждая из которых соответствует определённому эксперту: «Эксперт №1», «Эксперт №2», «Эксперт №3». После нажатия откроется окно (экспертное окно с оценками первого эксперта показано на рисунке 3.7), которое содержит матрицу парных сравнений. Для каждой пары можно выбрать свою оценку, при этом значения выше главной диагонали обратны значениям ниже главной диагонали, однако пользователь может изменять оценку в любом месте, программа определит правильное значение в другом месте, где должно быть по правилу обратное значение. Также для уменьшения вероятности того, что эксперт ошибся, в окне имеется текстовое поле, которое показывает отношение согласованности, которое не должно превышать значения 0,2. После изменения оценок, окно нужно просто закрыть по нажатию на иконку крестика в правом верхнем углу, так как все значения автоматически сохраняются.

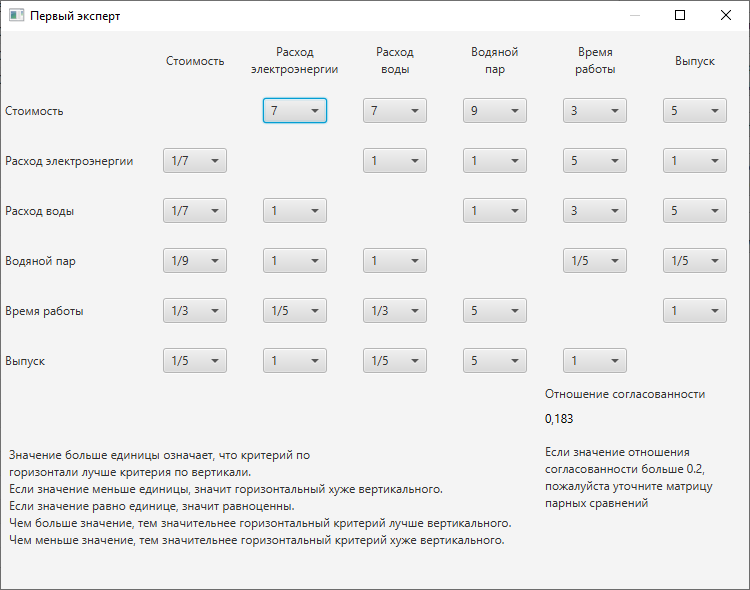


Рисунок 3.7 – Окно с оценками первого эксперта

Для расчёта весов критериев нужно нажать на кнопку «Метод экспертного анализа», после чего приложение рассчитает веса критериев на основании оценок трёх экспертов. На рисунке 3.8 показаны веса критериев для стандартных оценок экспертов, установленных в программе. Также оценки экспертов сбрасываются к стандартным при нажатии на кнопку «Сбросить всё».

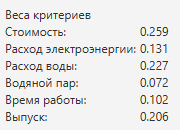


Рисунок 3.8 – Веса критериев, рассчитанные на основе стандартных экспертных оценок

Последним этапом решения задачи многокритериальной оптимизации является использование метода ЭЛЕКТРА. Результат показан на рисунке 3.9.

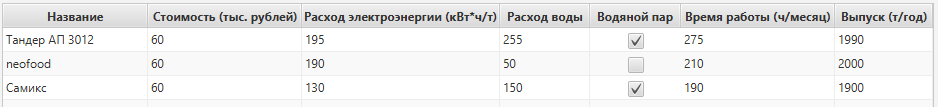


Рисунок 3.9 – Результат многокритериальной оптимизации на основе

стандартных значений

Методом ЭЛЕКТРА из оставшихся альтернатив выбираются лучшие на основании вышеприведённых весов критериев.

## **3.3 Результат верификации приложения**

Для верификации приложения следует провести однокритериальную и многокритериальную оптимизации с одинаковыми альтернативами, с одинаковыми ограничениями параметров.

Для однокритериальной оптимизации по условию выбран критерий расхода электроэнергии. Остальные критерии, которые заданы по условию задачи для линий производства зефира: стоимость, расход воды, необходимость водяного пара, время работы и выпускаемое количество продукции – не используются при решении задачи. Для ограничений также использованы значения заданные по условию задачи:

– выпуск зефира в количестве не менее 1000 тонн в год;

– месячное время работы линии (не более 300 часов);

– стоимость линии (не более 100 тысяч рублей);

– удельный расход электроэнергии (не более 200 киловатт в час на тонну).

На основе этих данных, решается задача однокритериальной оптимизации. Лучшей линией по результату расчёта является линия с названием «Самикс».

Для верификации многокритериальной оптимизации следует провести многокритериальную оптимизации на тех же данных и с теми же ограничениями, что и в случае однокритериальной оптимизации. Однако проблема заключается в том, что в случае многокритериальной оптимизации нельзя выбрать один главный критерий, всегда есть ещё в данном случае пять критериев, которые также влияют на расчёт, в результате чего ответ не всегда может совпасть. Для того, чтобы сделать минимальной вероятность несовпадения ответов, нужно в задаче многокритериальной оптимизации сделать так, чтобы необходимый критерий, который является критерием в задаче однокритериальной оптимизации, получил наивысшие оценки каждого эксперта (веса критериев после этого изменения показаны на рисунке 3.10).

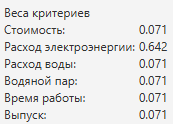


Рисунок 3.10 – Веса критериев после изменения оценок экспертами в пользу критерия расхода электроэнергии

Затем остаётся на основе этих весов решить задачу методом ЭЛЕКТРА. Результат показан на рисунке 3.11.

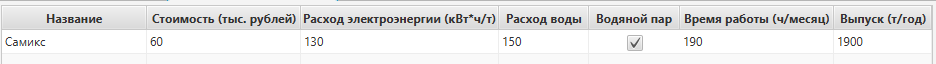


Рисунок 3.11 – Результат решения задачи многокритериальной оптимизации при тестировании по критерию расхода электроэнергии

Лучшей альтернативой, а именно линией по производству зефира, как и в случае однокритериальной оптимизации, является линия с названием «Самикс». Следовательно, есть все основания полагать, что оба метода оптимизации реализованы корректно и работают правильно.

# **ЗАКЛЮЧЕНИЕ**

Разработано программное обеспечение для однокритериальной и многокритериальной оптимизаций при проектировании участка производства зефира на языке программирования *Kotlin* с использованием библиотеки для создания графического пользовательского интерфейса *TornadoFX* (*JavaFX*). Особенностью приложения является кроссплатформенность, то есть возможность запустить программу на любой операционной системы, а также удобный и интуитивно понятный пользовательский интерфейс. Получившаяся программа полностью удовлетворяет всем поставленным требованиям и выполняет все необходимые задачи. Для проверки корректности работы программы проведена верификация.

Разработанная программа позволяет пользователю на основе вводимых данных используя метод полного перебора, метода экспертного анализа парных сравнений, метода ЭЛЕКТРА эффективно решить задачи по нахождению лучших линий производства зефира (альтернатив) среди списка представленных и использовать результат на практике, как дополнительное средство при выборе линий производства зефира. Также есть возможность использовать два вида оптимизации, что может дать дополнительную информацию пользователю при выборе.

В результате выполнения курсового проекта был приобретён опыт работы с однокритериальной и многокритериальной оптимизациями. Также была изучена предметная область – производство зефира в промышленных масштабах, изучены варианты его производства на разных линиях.

# **Список использованных источников**

1.  МНОГОКРИТЕРИАЛЬНАЯ ОПТИМИЗАЦИЯ. ПАРЕТО-ОПТИМАЛЬНЫЕ РЕШЕНИЯ – Студенческий научный форум. – Электрон. данные. – Режим доступа: https://scienceforum.ru/2021/article/2018024620. – Дата доступа: 21.11.2020.

2. Колемаева В. А. Методы оптимальных решений. Практикум : учеб. Пособие / В. А. Колемаева, В. И. Соловьёва. – М. : КНОРУС, 2017. – 194 с.

3. Кочкарова А. А. Методы принятия управленческих решений: количественный подход / А. А. Кочкарова. – М. : КНОРУС, 2016. – 146 с.

4. Лобачевский Н. И., Методы оптимизации в примерах и задачах / Н. И. Лобачевский, Бирюков Р. С. – Н. Новгород : Нижегородский госуниверситет, 2015. – 101 с.

5. Набатова Д. С. Математические и инструментальные методы поддержки принятия решений / Д. С. Набатова. – М. : Издательство Юрайт, 2015. – 292 с.

6. Тема 3.5. Многокритериальная оптимизация – Студопедия. – Студенческий научный форум. – Электрон. данные. – Режим доступа: https://studopedia.ru/4\_139903\_tema--mnogokriterialnaya-optimizatsiya.html. – Дата доступа: 22.11.2020.

# **ПРИЛОЖЕНИЕ А**

(обязательное)

**Чертёж схемы алгоритма**

# **ПРИЛОЖЕНИЕ Б**

(обязательное)

**Текст программы**

**Alternative.kt:**

package com.epicdima.oprcw

import com.beust.klaxon.Json

import com.beust.klaxon.Klaxon

import java.io.File

import java.util.\*

/\*\*

\* @author EpicDima

\*/

fun getAlternativesFromFile(filepath: String = DATA\_FILEPATH): List<Alternative> {

return Klaxon().parseArray(File(filepath))!!

}

data class Alternative(

var name: String = UUID.randomUUID().toString(), // Название

var cost: Int = 0, // Стоимость (тыс. рублей)

@Json(name = "power\_consumption")

var powerConsumption: Int = 0, // Расход электроэнергии (кВт\*ч/т)

@Json(name = "water\_consumption")

var waterConsumption: Int = 0, // Расход воды

var vapor: Boolean = false, // Необходимость водяного пара (да, нет)

@Json(name = "working\_time")

var workingTime: Int = 0, // Время работы в месяц

var output: Int = 0 // Выпуск (т в год)

) : Comparable<Alternative> {

val vaporNumber: Double

get() = if (vapor) 0.67 else 0.33

override fun compareTo(other: Alternative): Int {

return if (cost < other.cost && powerConsumption < other.powerConsumption

&& waterConsumption < other.waterConsumption && vaporNumber < other.vaporNumber

&& workingTime < other.workingTime && output > other.output

) {

1

} else if (cost > other.cost && powerConsumption > other.powerConsumption

&& waterConsumption > other.waterConsumption && vaporNumber > other.vaporNumber

&& workingTime > other.workingTime && output < other.output

) {

-1

} else {

0

}

}

}

data class DoubleAlternative(

var name: String = UUID.randomUUID().toString(), // Название

var cost: Double = 0.0, // Стоимость (тыс. рублей)

var powerConsumption: Double = 0.0, // Расход электроэнергии (кВт\*ч/т)

var waterConsumption: Double = 0.0, // Расход воды

var vapor: Double = 0.33, // Необходимость водяного пара (да, нет)

var workingTime: Double = 0.0, // Время работы в месяц

var output: Double = 0.0 // Выпуск (т в год)

)

**Constants.kt:**

package com.epicdima.oprcw

/\*\*

\* @author EpicDima

\*/

const val DATA\_FILEPATH = "E:\\Development\\Kotlin\\OPRCW\\data\\input.json"

const val CRITERIONS = 6

val marks = arrayOf("1/9", "1/7", "1/5", "1/3", "1", "3", "5", "7", "9")

val marksNumbers = doubleArrayOf(1.0 / 9.0, 1.0 / 7.0, 1.0 / 5.0, 1.0 / 3.0, 1.0, 3.0, 5.0, 7.0, 9.0)

**Application.kt:**

package com.epicdima.oprcw

import javafx.application.Application.launch

import tornadofx.App

/\*\*

\* @author EpicDima

\*/

fun main() {

launch(Application::class.java)

}

class Application : App() {

override val primaryView = MainView::class

}

**TableIntegerStringConverter.kt:**

package com.epicdima.oprcw

import javafx.util.StringConverter

/\*\*

\* @author EpicDima

\*/

class TableIntegerStringConverter : StringConverter<Int?>() {

override fun fromString(value: String): Int {

var s = value.trim().replace(Regex("[^\\d]"), "")

s = s.substring(0, 8.coerceAtMost(s.length))

return if (s.isEmpty()) {

0

} else {

val integer = s.toInt()

if (integer < 0) {

0

} else {

integer

}

}

}

override fun toString(value: Int?): String {

return value?.toString() ?: "0"

}

}

**ExpertView.kt:**

package com.epicdima.oprcw

import javafx.collections.FXCollections

import javafx.scene.control.ComboBox

import javafx.scene.control.Label

import javafx.scene.layout.GridPane

import javafx.scene.paint.Color

import tornadofx.Fragment

import tornadofx.onChange

import kotlin.math.pow

/\*\*

\* @author EpicDima

\*/

class ExpertView : Fragment() {

private val randomConsistency = 1.24

override val root: GridPane by fxml(javaClass.classLoader.getResourceAsStream("ExpertView.fxml")!!)

private val consistencyRatioLabel: Label by fxid()

private val name: String by param()

private val expert: Array<Array<String>> by param()

private val comboBoxes: MutableList<MutableList<ComboBox<String>?>> = mutableListOf()

init {

title = name

setConsistencyRatio()

for (i in 0 until CRITERIONS) {

comboBoxes.add(mutableListOf())

for (j in 0 until CRITERIONS) {

if (i != j) {

val comboBox = ComboBox<String>().apply {

items = FXCollections.observableList(marks.toList())

selectionModel.select(expert[i][j])

valueProperty().onChange {

expert[i][j] = it!!

val idx = marks.indexOf(expert[i][j])

expert[j][i] = when {

idx < 4 -> {

expert[i][j].removePrefix("1/")

}

idx > 4 -> {

"1/" + expert[i][j]

}

else -> {

"1"

}

}

comboBoxes[j][i]!!.selectionModel.select(expert[j][i])

setConsistencyRatio()

}

}

comboBoxes[i].add(comboBox)

root.add(comboBox, j + 1, i + 1)

} else {

comboBoxes[i].add(null)

}

}

}

}

private fun setConsistencyRatio() {

val consistencyRatio = calculateConsistencyRatio()

consistencyRatioLabel.text = String.format("%.3f", consistencyRatio)

if (consistencyRatio > 0.2) {

consistencyRatioLabel.textFill = Color.RED

} else {

consistencyRatioLabel.textFill = Color.BLACK

}

}

private fun calculateConsistencyRatio(): Double {

val columnSums = DoubleArray(CRITERIONS)

for (j in 0 until CRITERIONS) {

for (i in 0 until CRITERIONS) {

columnSums[j] += marksNumbers[marks.indexOf(expert[i][j])]

}

}

val weights = getWeights(expert)

var gamma = 0.0

for (j in 0 until CRITERIONS) {

gamma += columnSums[j] \* weights[j]

}

return ((gamma - CRITERIONS) / (CRITERIONS - 1)) / randomConsistency

}

private fun getWeights(expert: Array<Array<String>>): DoubleArray {

val criterionsCost = DoubleArray(CRITERIONS) { 1.0 }

for (i in 0 until CRITERIONS) {

for (j in 0 until CRITERIONS) {

criterionsCost[i] = criterionsCost[i] \* marksNumbers[marks.indexOf(expert[i][j])]

}

criterionsCost[i] = criterionsCost[i].pow(1.0 / CRITERIONS)

}

val sum = criterionsCost.sumByDouble { it }

return criterionsCost.map { it / sum }.toDoubleArray()

}

}

**MainView.kt:**

package com.epicdima.oprcw

import javafx.beans.value.ObservableValue

import javafx.collections.FXCollections

import javafx.scene.control.Label

import javafx.scene.control.TabPane

import javafx.scene.control.TableView

import javafx.scene.control.TextField

import javafx.util.StringConverter

import tornadofx.\*

import kotlin.math.pow

/\*\*

\* @author EpicDima

\*/

class MainView : View("MainView") {

override val root: TabPane by fxml(javaClass.classLoader.getResourceAsStream("MainView.fxml")!!)

private val firstTableView: TableView<Alternative> by fxid()

private val secondTableView: TableView<Alternative> by fxid()

private val firstOutputField: TextField by fxid()

private val firstWorkingTimeField: TextField by fxid()

private val firstPowerConsumptionField: TextField by fxid()

private val firstCostField: TextField by fxid()

private val secondOutputField: TextField by fxid()

private val secondWorkingTimeField: TextField by fxid()

private val secondPowerConsumptionField: TextField by fxid()

private val secondCostField: TextField by fxid()

private val costWeightLabel: Label by fxid()

private val powerConsumptionWeightLabel: Label by fxid()

private val waterConsumptionWeightLabel: Label by fxid()

private val vaporWeightLabel: Label by fxid()

private val workingTimeWeightLabel: Label by fxid()

private val outputWeightLabel: Label by fxid()

private var firstList = FXCollections.observableArrayList<Alternative>()

private var secondList = FXCollections.observableArrayList<Alternative>()

private var firstExpert: Array<Array<String>> = Array(CRITERIONS) { Array(CRITERIONS) { "" } }

private var secondExpert: Array<Array<String>> = Array(CRITERIONS) { Array(CRITERIONS) { "" } }

private var thirdExpert: Array<Array<String>> = Array(CRITERIONS) { Array(CRITERIONS) { "" } }

private lateinit var criterionsWeights: DoubleArray

init {

title = "OPRCW"

loadFirstFromFile()

loadSecondFromFile()

firstTableView.apply {

items = firstList

setColumns()

columns.forEach {

it.isSortable = false

}

contextmenu {

item("Удалить").action {

val selected = selectedItem

if (selected != null) {

firstList.remove(selected)

selectionModel.clearSelection()

}

}

}

}

secondTableView.apply {

items = secondList

setColumns()

columns.forEach {

it.isSortable = false

}

contextmenu {

item("Удалить").action {

val selected = selectedItem

if (selected != null) {

secondList.remove(selected)

selectionModel.clearSelection()

}

}

}

}

firstOutputField.textProperty().addListener(createNumberTextListener(firstOutputField))

firstWorkingTimeField.textProperty().addListener(createNumberTextListener(firstWorkingTimeField))

firstPowerConsumptionField.textProperty().addListener(createNumberTextListener(firstPowerConsumptionField))

firstCostField.textProperty().addListener(createNumberTextListener(firstCostField))

secondOutputField.textProperty().addListener(createNumberTextListener(secondOutputField))

secondWorkingTimeField.textProperty().addListener(createNumberTextListener(secondWorkingTimeField))

secondPowerConsumptionField.textProperty().addListener(createNumberTextListener(secondPowerConsumptionField))

secondCostField.textProperty().addListener(createNumberTextListener(secondCostField))

firstReset()

secondReset()

}

private fun createNumberTextListener(field: TextField) = { \_: ObservableValue<out String>, \_: String, new: String ->

if (!new.matches(Regex("\\d\*"))) {

var s = new.replace(Regex("[^\\d]"), "")

s = s.substring(0, 8.coerceAtMost(s.length))

if (s.isEmpty()) {

field.text = "0"

} else {

field.text = s

}

}

}

private fun TableView<Alternative>.setColumns() {

column("Название", Alternative::name).makeEditable()

column(

"Стоимость (тыс. рублей)",

Alternative::cost

).makeEditable(TableIntegerStringConverter() as StringConverter<Int>)

column("Расход электроэнергии (кВт\*ч/т)", Alternative::powerConsumption).makeEditable(

TableIntegerStringConverter() as StringConverter<Int>

)

column(

"Расход воды",

Alternative::waterConsumption

).makeEditable(TableIntegerStringConverter() as StringConverter<Int>)

column("Водяной пар", Alternative::vapor).makeEditable()

column(

"Время работы (ч/месяц)",

Alternative::workingTime

).makeEditable(TableIntegerStringConverter() as StringConverter<Int>)

column(

"Выпуск (т/год)",

Alternative::output

).makeEditable(TableIntegerStringConverter() as StringConverter<Int>)

}

private fun loadFirstFromFile() {

firstList.setAll(getAlternativesFromFile())

}

private fun loadSecondFromFile() {

secondList.setAll(getAlternativesFromFile())

}

fun firstAdd() {

firstReset()

firstList.add(Alternative())

}

fun firstReset() {

loadFirstFromFile()

firstOutputField.text = "1000"

firstWorkingTimeField.text = "300"

firstPowerConsumptionField.text = "200"

firstCostField.text = "100"

}

fun firstCalculate() {

filterList(

firstList,

firstOutputField.text.toInt(),

firstWorkingTimeField.text.toInt(),

firstPowerConsumptionField.text.toInt(),

firstCostField.text.toInt()

)

firstList.setAll(firstList.sorted { o1, o2 -> o1.powerConsumption - o2.powerConsumption })

}

fun secondAdd() {

secondReset()

secondList.add(Alternative())

}

fun secondReset() {

loadSecondFromFile()

secondOutputField.text = "1000"

secondWorkingTimeField.text = "300"

secondPowerConsumptionField.text = "200"

secondCostField.text = "100"

resetFirstExpert()

resetSecondExpert()

resetThirdExpert()

costWeightLabel.text = "0"

powerConsumptionWeightLabel.text = "0"

waterConsumptionWeightLabel.text = "0"

vaporWeightLabel.text = "0"

workingTimeWeightLabel.text = "0"

outputWeightLabel.text = "0"

criterionsWeights = DoubleArray(0)

}

private fun filterList(

list: MutableList<Alternative>,

output: Int,

workingTime: Int,

powerConsumption: Int,

cost: Int

) {

list.removeIf {

it.output < output

|| it.workingTime > workingTime

|| it.powerConsumption > powerConsumption

|| it.cost > cost

}

}

fun removeAlternatives() {

filterList(

secondList,

secondOutputField.text.toInt(),

secondWorkingTimeField.text.toInt(),

secondPowerConsumptionField.text.toInt(),

secondCostField.text.toInt()

)

}

fun expertAnalysis() {

val firstExpertWeight = getWeights(firstExpert)

val secondExpertWeight = getWeights(secondExpert)

val thirdExpertWeight = getWeights(thirdExpert)

criterionsWeights =

DoubleArray(CRITERIONS) { (firstExpertWeight[it] + secondExpertWeight[it] + thirdExpertWeight[it]) / 3.0 }

costWeightLabel.text = criterionsWeights[0].toString().substring(0, 5)

powerConsumptionWeightLabel.text = criterionsWeights[1].toString().substring(0, 5)

waterConsumptionWeightLabel.text = criterionsWeights[2].toString().substring(0, 5)

vaporWeightLabel.text = criterionsWeights[3].toString().substring(0, 5)

workingTimeWeightLabel.text = criterionsWeights[4].toString().substring(0, 5)

outputWeightLabel.text = criterionsWeights[5].toString().substring(0, 5)

}

private fun getWeights(expert: Array<Array<String>>): DoubleArray {

val criterionsCost = DoubleArray(CRITERIONS) { 1.0 }

for (i in 0 until CRITERIONS) {

for (j in 0 until CRITERIONS) {

criterionsCost[i] = criterionsCost[i] \* marksNumbers[marks.indexOf(expert[i][j])]

}

criterionsCost[i] = criterionsCost[i].pow(1.0 / CRITERIONS)

}

val sum = criterionsCost.sumByDouble { it }

return criterionsCost.map { it / sum }.toDoubleArray()

}

fun pareto() {

var i = 0

var j = 1

while (true) {

val result = secondList[i].compareTo(secondList[j])

if (result < 0) {

secondList.remove(i, i + 1)

} else if (result > 0) {

secondList.remove(j, j + 1)

}

if (j < secondList.size - 1) {

j += 1

} else if (i < secondList.size - 2) {

i += 1

j = i + 1

} else {

break

}

}

}

fun elektra() {

val result = mutableListOf<Alternative>()

val normalizedAlternatives = normalizeSecondList()

val criterionSums = calculateCriterionSums(normalizedAlternatives)

for (i in 0 until normalizedAlternatives.size - 1) {

for (j in i + 1 until normalizedAlternatives.size) {

val agreement = calculateAgreement(normalizedAlternatives[i], normalizedAlternatives[j])

val disagreement =

calculateDisagreement(normalizedAlternatives[i], normalizedAlternatives[j], criterionSums)

if (agreement > 0.5 && disagreement < 0.5) {

result.add(secondList.first { it.name == normalizedAlternatives[i].name })

}

}

}

secondList.setAll(result.distinct())

}

private fun normalizeSecondList(): MutableList<DoubleAlternative> {

val costMin = secondList.minOf { it.cost }

val costMax = secondList.maxOf { it.cost }

val powerConsumptionMin = secondList.minOf { it.powerConsumption }

val powerConsumptionMax = secondList.maxOf { it.powerConsumption }

val waterConsumptionMin = secondList.minOf { it.waterConsumption }

val waterConsumptionMax = secondList.maxOf { it.waterConsumption }

val workingTimeMin = secondList.minOf { it.workingTime }

val workingTimeMax = secondList.maxOf { it.workingTime }

val outputMin = secondList.minOf { it.output }

val outputMax = secondList.maxOf { it.output }

return secondList.map {

DoubleAlternative(

it.name,

(it.cost.toDouble() - costMin) / (costMax - costMin),

(it.powerConsumption.toDouble() - powerConsumptionMin) / (powerConsumptionMax - powerConsumptionMin),

(it.waterConsumption.toDouble() - waterConsumptionMin) / (waterConsumptionMax - waterConsumptionMin),

it.vaporNumber,

(it.workingTime.toDouble() - workingTimeMin) / (workingTimeMax - workingTimeMin),

(it.output.toDouble() - outputMin) / (outputMax - outputMin)

)

}.toMutableList()

}

private fun calculateCriterionSums(alternatives: List<DoubleAlternative>): DoubleArray {

return doubleArrayOf(

alternatives.sumOf { it.cost },

alternatives.sumOf { it.powerConsumption },

alternatives.sumOf { it.waterConsumption },

alternatives.sumOf { it.vapor },

alternatives.sumOf { it.workingTime },

alternatives.sumOf { it.output },

)

}

private fun calculateAgreement(first: DoubleAlternative, second: DoubleAlternative): Double {

var result = 0.0

if (first.cost <= second.cost) {

result += criterionsWeights[0]

}

if (first.powerConsumption <= second.powerConsumption) {

result += criterionsWeights[1]

}

if (first.waterConsumption <= second.waterConsumption) {

result += criterionsWeights[2]

}

if (first.vapor <= second.vapor) {

result += criterionsWeights[3]

}

if (first.workingTime <= second.workingTime) {

result += criterionsWeights[4]

}

if (first.output >= second.output) {

result += criterionsWeights[5]

}

return result

}

private fun calculateDisagreement(

first: DoubleAlternative,

second: DoubleAlternative,

criterionSums: DoubleArray

): Double {

val marks = mutableListOf<Double>()

if (first.cost >= second.cost) {

marks.add((second.cost - first.cost) / criterionSums[0])

}

if (first.powerConsumption >= second.powerConsumption) {

marks.add((second.powerConsumption - first.powerConsumption) / criterionSums[1])

}

if (first.waterConsumption >= second.waterConsumption) {

marks.add((second.waterConsumption - first.waterConsumption) / criterionSums[2])

}

if (first.vapor >= second.vapor) {

marks.add((second.vapor - first.vapor) / criterionSums[3])

}

if (first.workingTime >= second.workingTime) {

marks.add((second.workingTime - first.workingTime) / criterionSums[4])

}

if (first.output <= second.output) {

marks.add((second.output - first.output) / criterionSums[5])

}

return marks.maxOfOrNull { it } ?: 0.0

}

private fun resetFirstExpert() {

with(marks) {

firstExpert = arrayOf(

arrayOf(get(4), get(7), get(7), get(8), get(5), get(6)),

arrayOf(get(4), get(4), get(4), get(4), get(6), get(4)),

arrayOf(get(4), get(4), get(4), get(4), get(5), get(6)),

arrayOf(get(4), get(4), get(4), get(4), get(2), get(2)),

arrayOf(get(4), get(4), get(4), get(4), get(4), get(4)),

arrayOf(get(4), get(4), get(4), get(4), get(4), get(4)),

)

}

completeExpert(firstExpert)

}

private fun resetSecondExpert() {

with(marks) {

secondExpert = arrayOf(

arrayOf(get(4), get(3), get(4), get(4), get(2), get(0)),

arrayOf(get(4), get(4), get(7), get(5), get(2), get(2)),

arrayOf(get(4), get(4), get(4), get(3), get(3), get(2)),

arrayOf(get(4), get(4), get(4), get(4), get(4), get(2)),

arrayOf(get(4), get(4), get(4), get(4), get(4), get(2)),

arrayOf(get(4), get(4), get(4), get(4), get(4), get(4)),

)

}

completeExpert(secondExpert)

}

private fun resetThirdExpert() {

with(marks) {

thirdExpert = arrayOf(

arrayOf(get(4), get(5), get(2), get(6), get(6), get(5)),

arrayOf(get(4), get(4), get(2), get(5), get(5), get(8)),

arrayOf(get(4), get(4), get(4), get(8), get(8), get(6)),

arrayOf(get(4), get(4), get(4), get(4), get(7), get(5)),

arrayOf(get(4), get(4), get(4), get(4), get(4), get(5)),

arrayOf(get(4), get(4), get(4), get(4), get(4), get(4)),

)

}

completeExpert(thirdExpert)

}

private fun completeExpert(expert: Array<Array<String>>) {

for (i in 0 until CRITERIONS) {

for (j in 0 until i) {

val idx = marks.indexOf(expert[j][i])

expert[i][j] = when {

idx < 4 -> {

expert[j][i].removePrefix("1/")

}

idx > 4 -> {

"1/" + expert[j][i]

}

else -> {

"1"

}

}

}

}

}

fun firstExpert() {

find<ExpertView>(mapOf("name" to "Первый эксперт", "expert" to firstExpert)).openModal()

}

fun secondExpert() {

find<ExpertView>(mapOf("name" to "Второй эксперт", "expert" to secondExpert)).openModal()

}

fun thirdExpert() {

find<ExpertView>(mapOf("name" to "Третий эксперт", "expert" to thirdExpert)).openModal()

}

}

**ExpertView.fxml:**

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.layout.ColumnConstraints?>

<?import javafx.scene.layout.GridPane?>

<?import javafx.scene.layout.RowConstraints?>

<?import javafx.scene.layout.VBox?>

<GridPane fx:id="grid" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1">

<columnConstraints>

<ColumnConstraints hgrow="SOMETIMES" minWidth="-Infinity" prefWidth="140.0" />

<ColumnConstraints halignment="CENTER" hgrow="SOMETIMES" minWidth="-Infinity" prefWidth="100.0" />

<ColumnConstraints halignment="CENTER" hgrow="SOMETIMES" minWidth="-Infinity" prefWidth="100.0" />

<ColumnConstraints halignment="CENTER" hgrow="SOMETIMES" minWidth="-Infinity" prefWidth="100.0" />

<ColumnConstraints halignment="CENTER" hgrow="SOMETIMES" minWidth="-Infinity" prefWidth="100.0" />

<ColumnConstraints halignment="CENTER" hgrow="SOMETIMES" minWidth="-Infinity" prefWidth="100.0" />

<ColumnConstraints halignment="CENTER" hgrow="SOMETIMES" minWidth="-Infinity" prefWidth="100.0" />

</columnConstraints>

<rowConstraints>

<RowConstraints minHeight="-Infinity" prefHeight="50.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="-Infinity" prefHeight="50.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="-Infinity" prefHeight="50.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="-Infinity" prefHeight="50.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="-Infinity" prefHeight="50.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="-Infinity" prefHeight="50.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="-Infinity" prefHeight="50.0" vgrow="SOMETIMES" />

<RowConstraints vgrow="SOMETIMES" />

</rowConstraints>

<children>

<Label text="Стоимость" textAlignment="CENTER" GridPane.columnIndex="1" />

<Label text="Расход&#10;электроэнергии" textAlignment="CENTER" GridPane.columnIndex="2" />

<Label text="Расход&#10;воды" textAlignment="CENTER" GridPane.columnIndex="3" />

<Label text="Водяной&#10;пар" textAlignment="CENTER" GridPane.columnIndex="4" />

<Label text="Время&#10;работы" textAlignment="CENTER" GridPane.columnIndex="5" />

<Label text="Выпуск" textAlignment="CENTER" GridPane.columnIndex="6" />

<Label text="Стоимость" GridPane.rowIndex="1" />

<Label text="Расход электроэнергии" GridPane.rowIndex="2" />

<Label text="Расход воды" GridPane.rowIndex="3" />

<Label text="Водяной пар" GridPane.rowIndex="4" />

<Label text="Время работы" GridPane.rowIndex="5" />

<Label text="Выпуск" GridPane.rowIndex="6" />

<Label text="Значение больше единицы означает, что критерий по&#10;горизонтали лучше критерия по вертикали. &#10;Если значение меньше единицы, значит горизонтальный хуже вертикального. &#10;Если значение равно единице, значит равноценны.&#10;Чем больше значение, тем значительнее горизонтальный критерий лучше вертикального.&#10;Чем меньше значение, тем значительнее горизонтальный критерий хуже вертикального." GridPane.columnSpan="5" GridPane.rowIndex="7">

<GridPane.margin>

<Insets top="24.0" />

</GridPane.margin>

<padding>

<Insets bottom="4.0" left="4.0" right="4.0" top="4.0" />

</padding>

</Label>

<VBox prefHeight="200.0" prefWidth="100.0" spacing="8.0" GridPane.columnIndex="5" GridPane.columnSpan="2" GridPane.rowIndex="7">

<children>

<Label text="Отношение согласованности" />

<Label fx:id="consistencyRatioLabel" layoutX="10.0" layoutY="27.0" text="0" />

<Label fx:id="consistencyRatioMessageLabel" text="Если значение отношения &#10;согласованности больше 0.2,&#10;пожалуйста уточните матрицу &#10;парных сравнений&#10;">

<padding>

<Insets top="8.0" />

</padding>

</Label>

</children>

</VBox>

</children>

<padding>

<Insets bottom="4.0" left="4.0" right="4.0" top="4.0" />

</padding>

</GridPane>

**MainView.fxml:**

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>

<?import javafx.scene.control.Button?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.control.Tab?>

<?import javafx.scene.control.TabPane?>

<?import javafx.scene.control.TableView?>

<?import javafx.scene.control.TextField?>

<?import javafx.scene.layout.HBox?>

<?import javafx.scene.layout.VBox?>

<TabPane prefHeight="500.0" prefWidth="760.0" tabClosingPolicy="UNAVAILABLE" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1">

<tabs>

<Tab text="Однокритериальная задача">

<content>

<VBox>

<children>

<HBox spacing="12.0" VBox.vgrow="ALWAYS">

<children>

<TableView fx:id="firstTableView" HBox.hgrow="ALWAYS" />

<VBox alignment="TOP\_CENTER" prefWidth="200.0" spacing="16.0">

<children>

<Button fx:id="firstAddButton" mnemonicParsing="false" onAction="#firstAdd" prefWidth="150.0" text="Добавить альтернативу" />

<Button fx:id="firstResetButton" mnemonicParsing="false" onAction="#firstReset" prefWidth="150.0" text="Сбросить всё" />

<Button fx:id="firstCalculateButton" mnemonicParsing="false" onAction="#firstCalculate" prefWidth="150.0" text="Рассчитать" />

</children>

</VBox>

</children>

</HBox>

<VBox spacing="4.0">

<children>

<Label text="Ограничения" />

<HBox spacing="12.0">

<children>

<VBox prefWidth="200.0">

<children>

<Label text="Выпуск (т/год) (&gt;)" />

<TextField fx:id="firstOutputField" />

</children>

</VBox>

<VBox prefWidth="200.0">

<children>

<Label text="Время работы (ч/месяц) (&lt;)" />

<TextField fx:id="firstWorkingTimeField" />

</children>

</VBox>

<VBox prefWidth="200.0">

<children>

<Label text="Расход электроэнергии (кВт\*ч/т) (&lt;)" />

<TextField fx:id="firstPowerConsumptionField" />

</children>

</VBox>

<VBox prefWidth="200.0">

<children>

<Label text="Стоимость (тыс. рублей) (&gt;)" />

<TextField fx:id="firstCostField" />

</children>

</VBox>

</children>

</HBox>

</children>

<VBox.margin>

<Insets top="8.0" />

</VBox.margin>

</VBox>

</children>

<padding>

<Insets bottom="4.0" left="4.0" right="4.0" top="4.0" />

</padding>

</VBox>

</content>

</Tab>

<Tab text="Многокритериальная задача">

<content>

<VBox>

<children>

<HBox spacing="28.0" VBox.vgrow="ALWAYS">

<children>

<TableView fx:id="secondTableView" editable="true" HBox.hgrow="ALWAYS" />

<VBox minWidth="-Infinity" prefWidth="190.0" spacing="16.0">

<children>

<Button fx:id="secondAddButton" mnemonicParsing="false" onAction="#secondAdd" prefWidth="200.0" text="Добавить альтернативу" textAlignment="CENTER" />

<Button fx:id="secondResetButton" mnemonicParsing="false" onAction="#secondReset" prefHeight="0.0" prefWidth="200.0" text="Сбросить всё" />

<Button fx:id="removeAlternativesButton" mnemonicParsing="false" onAction="#removeAlternatives" prefHeight="43.0" prefWidth="200.0" text="Убрать альтернативы не&#10;соответствующие условиям" textAlignment="CENTER" />

<Button fx:id="expertAnalysisButton" mnemonicParsing="false" onAction="#expertAnalysis" prefHeight="0.0" prefWidth="200.0" text="Метод экспертного анализа" textAlignment="CENTER" />

<Button fx:id="paretoButton" mnemonicParsing="false" onAction="#pareto" prefHeight="43.0" prefWidth="200.0" text="Отбор бесперспективных&#10;альтернатив (метод Парето)" textAlignment="CENTER" />

<Button fx:id="elektraButton" mnemonicParsing="false" onAction="#elektra" prefHeight="41.0" prefWidth="200.0" text="Выбрать лучшие альтернативы&#10;(метод Электра)" textAlignment="CENTER" />

</children>

</VBox>

<VBox alignment="TOP\_CENTER" minWidth="-Infinity" prefWidth="180.0" spacing="10.0">

<children>

<Button fx:id="firstExpertButton" mnemonicParsing="false" onAction="#firstExpert" text="Эксперт №1" />

<Button fx:id="secondExpertButton" mnemonicParsing="false" onAction="#secondExpert" text="Эксперт №2" />

<Button fx:id="thirdExpertButton" mnemonicParsing="false" onAction="#thirdExpert" text="Эксперт №3" />

<VBox>

<children>

<Label text="Веса критериев" />

<HBox>

<children>

<Label prefWidth="135.0" text="Стоимость:" />

<Label fx:id="costWeightLabel" text="0" />

</children>

</HBox>

<HBox>

<children>

<Label prefWidth="135.0" text="Расход электроэнергии:" />

<Label fx:id="powerConsumptionWeightLabel" text="0" />

</children>

</HBox>

<HBox>

<children>

<Label prefWidth="135.0" text="Расход воды:" />

<Label fx:id="waterConsumptionWeightLabel" text="0" />

</children>

</HBox>

<HBox>

<children>

<Label prefWidth="135.0" text="Водяной пар:" />

<Label fx:id="vaporWeightLabel" text="0" />

</children>

</HBox>

<HBox>

<children>

<Label prefWidth="135.0" text="Время работы:" />

<Label fx:id="workingTimeWeightLabel" text="0" />

</children>

</HBox>

<HBox>

<children>

<Label fx:id="outputWeightLabel" prefWidth="135.0" text="Выпуск:" />

<Label fx:id="outputWeightLabel" text="0" />

</children>

</HBox>

</children>

</VBox>

</children>

</VBox>

</children>

</HBox>

<VBox spacing="4.0">

<children>

<Label text="Ограничения" />

<HBox spacing="12.0">

<children>

<VBox prefWidth="200.0">

<children>

<Label text="Выпуск (т/год) (&gt;)" />

<TextField fx:id="secondOutputField" />

</children>

</VBox>

<VBox prefWidth="200.0">

<children>

<Label text="Время работы (ч/месяц) (&lt;)" />

<TextField fx:id="secondWorkingTimeField" />

</children>

</VBox>

<VBox prefWidth="200.0">

<children>

<Label text="Расход электроэнергии (кВт\*ч/т) (&lt;)" />

<TextField fx:id="secondPowerConsumptionField" />

</children>

</VBox>

<VBox prefWidth="200.0">

<children>

<Label text="Стоимость (тыс. рублей) (&gt;)" />

<TextField fx:id="secondCostField" />

</children>

</VBox>

</children>

</HBox>

</children>

<VBox.margin>

<Insets top="8.0" />

</VBox.margin>

</VBox>

</children>

<padding>

<Insets bottom="4.0" left="4.0" right="4.0" top="4.0" />

</padding>

</VBox>

</content></Tab>

</tabs>

</TabPane>