

Министерство науки и высшего образования Российской Федерации  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО  
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА/GRADUATION THESIS

Оптимизация синтеза в САПР специализированных вычислителей с помощью  
машинного обучения

**Автор/ Author**

Бураков Илья Алексеевич

**Направленность (профиль) образовательной программы/Major**

Программно-информационные системы 2017

**Квалификация/ Degree level**

Бакалавр

**Руководитель ВКР/ Thesis supervisor**

Пенской Александр Владимирович, кандидат технических наук, Университет ИТМО,  
факультет программной инженерии и компьютерной техники, доцент (квалификационная  
категория "ординарный доцент")

**Группа/Group**

P3417

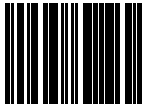
**Факультет/институт/кластер/ Faculty/Institute/Cluster**

факультет программной инженерии и компьютерной техники

**Направление подготовки/ Subject area**

09.03.04 Программная инженерия

Обучающийся/Student

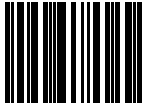
Документ подписан	
Бураков Илья Алексеевич	
29.05.2021	

(эл. подпись/ signature)

Бураков Илья  
Алексеевич

(Фамилия И.О./ name  
and surname)

Руководитель ВКР/  
Thesis supervisor

Документ подписан	
Пенской Александр Владимирович	
29.05.2021	

(эл. подпись/ signature)

Пенской  
Александр  
Владимирович

(Фамилия И.О./ name  
and surname)

**Министерство науки и высшего образования Российской Федерации  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО  
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /  
OBJECTIVES FOR A GRADUATION THESIS**

**Обучающийся / Student** Бураков Илья Алексеевич

**Группа/Group** P3417

**Факультет/институт/кластер/ Faculty/Institute/Cluster** факультет программной инженерии и компьютерной техники

**Квалификация/ Degree level** Бакалавр

**Направление подготовки/ Subject area** 09.03.04 Программная инженерия

**Направленность (профиль) образовательной программы/Major** Программно-информационные системы 2017

**Специализация/ Specialization**

**Тема ВКР/ Thesis topic** Оптимизация синтеза в САПР специализированных вычислителей с помощью машинного обучения

**Руководитель ВКР/ Thesis supervisor** Пенской Александр Владимирович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

**Срок сдачи студентом законченной работы до / Deadline for submission of complete thesis** 31.05.2021

**Техническое задание и исходные данные к работе/ Requirements and premise for the thesis**

Изучить существующие реализации метода синтеза в системе автоматического проектирования специализированных процессоров. Повысить эффективность процесса синтеза за счёт статистически обоснованного эвристического алгоритма неполного перебора с применением машинного обучения. Исследовать эффективность разработанного решения, считая критериями длительность процесса синтеза и время исполнения управляющей программы.

**Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)/ Content of the thesis (list of key issues)**

- 1) Введение. Актуальность темы. Постановка проблемы. Цели и задачи исследования.
- 2) Анализ существующих реализаций метода синтеза. Формализация задачи для алгоритмов машинного обучения.
- 3) Анализ существующих применений машинного обучения к аналогичным проблемам.
- 4) Генерация и подготовка исходного набора данных. Инженерия признаков.

- 5) Разработка модели машинного обучения. Обучение и тестирование.
- 6) Исследование эффективности разработанного решения. Сравнение с ранее реализованными методами синтеза.
- 7) Заключение. Выводы. Анализ возможных направлений для дальнейших исследований.

**Перечень графического материала (с указанием обязательного материала) / List of graphic materials (with a list of required material)**

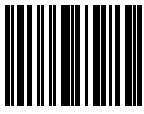
**Исходные материалы и пособия / Source materials and publications**

- [1] Nielsen, M. Neural Networks and Deep Learning, Determination Press, 2015.
- [2] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. Tensorflow: A system for large-scale machine learning. 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16) (pp. 265-283), 2016.
- [3] Пенской А.В., Платунов А.Е., Ключев А.О., Горбачев Я.Г., Яналов Р.И. Система высокоуровневого синтеза на основе гибридной реконфигурируемой микроархитектуры, 2019.

**Дата выдачи задания/ Objectives issued on 23.04.2021**

**СОГЛАСОВАНО / AGREED:**

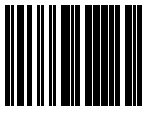
Руководитель ВКР/  
Thesis supervisor

Документ подписан	
Пенской Александр Владимирович	
23.04.2021	

Пенской  
Александр  
Владимирович

(эл. подпись)

Задание принял к  
исполнению/ Objectives  
assumed by

Документ подписан	
Бураков Илья Алексеевич	
23.04.2021	

Бураков Илья  
Алексеевич

(эл. подпись)

Руководитель ОП/ Head  
of educational program

Документ подписан	
Муромцев Дмитрий Ильич	
20.05.2021	

Муромцев  
Дмитрий Ильич

(эл. подпись)

**Министерство науки и высшего образования Российской Федерации  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО  
ITMO University**

**АННОТАЦИЯ  
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ /  
SUMMARY OF A GRADUATION THESIS**

**Обучающийся/ Student**

Бураков Илья Алексеевич

**Наименование темы ВКР / Title of the thesis**

Оптимизация синтеза в САПР специализированных вычислителей с помощью машинного обучения

**Наименование организации, где выполнена ВКР/ Name of organization**

Университет ИТМО

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ/  
DESCRIPTION OF THE GRADUATION THESIS**

**1. Цель исследования / Research objective**

Повысить эффективность процесса синтеза специализированных процессоров за счёт статистически обоснованного эвристического алгоритма неполного перебора с применением машинного обучения.

**2. Задачи, решаемые в ВКР / Research tasks**

Изучение существующих применений машинного обучения к аналогичным проблемам. Формализация предметной области для машинного обучения. Разработка алгоритма процесса синтеза с помощью машинного обучения. Исследование эффективности разработанного решения.

**3. Краткая характеристика полученных результатов / Short summary of results/conclusions**

Предложенный метод позволяет получить статистически обоснованную целевую функцию, с помощью которой реализуется эффективный алгоритм синтеза расписания управляющей программы. Полученные результаты позволяют считать работу успешной проверкой концепции и мотивируют продолжать исследования в этой области по определённым в работе направлениям.

**4. Наличие публикаций по теме выпускной работы/ Have you produced any publications on the topic of the thesis**

- 1 Бураков И.А. Оптимизация синтеза в САПР специализированных вычислителей с помощью машинного обучения//Сборник тезисов докладов конгресса молодых ученых. Электронное издание. Режим доступа: <https://kmu.itmo.ru/digests/article/7086>, своб. - 2021 (Тезисы)

**5. Наличие выступлений на конференциях по теме выпускной работы/ Have you**

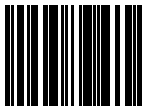
**produced any conference reports on the topic of the thesis**

- 1 Юбилейный X Конгресс молодых ученых, 14.04.2021 - 17.04.2021 (Конгресс, статус - всероссийский)

**6. Полученные гранты, при выполнении работы/ Grants received while working on the thesis**

**7. Дополнительные сведения/ Additional information**

Обучающийся/Student

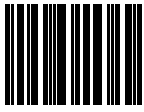
Документ подписан	
Бураков Илья Алексеевич	
29.05.2021	

(эл. подпись/ signature)

Бураков Илья  
Алексеевич

(Фамилия И.О./ name  
and surname)

Руководитель ВКР/  
Thesis supervisor

Документ подписан	
Пенской Александр Владимирович	
29.05.2021	

(эл. подпись/ signature)

Пенской  
Александр  
Владимирович

(Фамилия И.О./ name  
and surname)

## СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	7
ВВЕДЕНИЕ.....	8
1 ОБЗОР СУЩЕСТВУЮЩИХ РЕАЛИЗАЦИЙ МЕТОДОВ СИНТЕЗА.....	11
1.1 Метод синтеза в системе NITTA .....	11
1.2 Применение машинного обучения к деревьям решений.....	15
1.3 Выводы.....	18
2 ПРОЕКТИРОВАНИЕ РЕШЕНИЯ НА БАЗЕ МАШИННОГО ОБУЧЕНИЯ.....	19
2.1 Формализация задачи для алгоритмов машинного обучения.....	19
2.2 Описание модели .....	26
2.3 Выводы.....	27
3 РЕАЛИЗАЦИЯ РЕШЕНИЯ НА БАЗЕ МАШИННОГО ОБУЧЕНИЯ.....	29
3.1 Сбор тренировочных данных .....	29
3.2 Подготовка данных .....	33
3.3 Реализация, обучение и тестирование модели.....	35
3.4 Выводы.....	37
4 ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ РЕШЕНИЯ .....	38
4.1 Определение метода .....	38
4.2 Проведение исследования.....	39
4.3 Выводы.....	42
ЗАКЛЮЧЕНИЕ.....	43
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	45
ПРИЛОЖЕНИЕ А (РЕАЛИЗАЦИЯ СБОРЩИКА ДАННЫХ) .....	47
ПРИЛОЖЕНИЕ Б (РЕАЛИЗАЦИЯ МОДЕЛИ МАШИННОГО ОБУЧЕНИЯ).....	51
ПРИЛОЖЕНИЕ В (РЕАЛИЗАЦИЯ ИССЛЕДОВАНИЯ ЭФФЕКТИВНОСТИ) ....	53

## СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

САПР – система автоматического проектирования

ПЛИС – программируемая логическая интегральная схема

RTL – Register-Transfer Level – уровень передач данных между регистрами в контексте разработки интегральных схем, на котором алгоритм работы схемы задаётся последовательностью операций над данными при их пересылке между регистрами

HLS – High Level Synthesis – синтез RTL-спецификации алгоритма, заданного на высокоуровневом языке, в САПР специализированных вычислителей

МО – машинное обучение

MCTS – Monte Carlo Tree Search – метод Монте-Карло для поиска в графе

OHE – One-Hot Encoding – кодирование с использованием индивидуальных переменных для каждой категории в контексте отображения категориальных данных в численный вектор

ReLU – Rectified Linear Unit – блок линейной ректификации, одна из возможных активационных функций в искусственных нейронных сетях

MSE – Mean Squared Error – среднеквадратическая ошибка

MAE – Mean Absolute Error – средняя абсолютная величина ошибки

## ВВЕДЕНИЕ

Тема исследования актуальна в контексте разработки системы автоматического проектирования (САПР) специализированных процессоров. Такие системы используются с целью автоматизации проектирования специализированных процессоров для определённого класса задач, требующих детерминированности времени работы алгоритма, высокой скорости обработки данных и низкого энергопотребления, к примеру обработка сигналов, автоматизированное управление физическими объектами и процессами, моделирование в реальном времени и другие подобные задачи [1]. САПР специализированных процессоров позволяют существенно сократить время разработки сложных вычислительных систем, включающих в себя подобные решения, особенно когда они реализуют часто изменяемые алгоритмы, используя реконфигурируемость программируемых логических интегральных схем (ПЛИС) [2].

Стандартным способом разработки специализированного вычислителя на ПЛИС является описание реализуемого алгоритма на уровне регистровых передач (Register-Transfer Level, RTL) с использованием специализированных языков (Verilog, VHDL) [1]. Такой подход требует высоких трудозатрат и привлечения узких специалистов, особенно если в прикладной алгоритм регулярно вносятся корректировки. Альтернативным подходом является генерация специализированных вычислителей средствами систем высокоуровневого синтеза (High Level Synthesis, HLS) или применение САПР специализированных вычислителей, позволяющих использовать языки высокого уровня для описания прикладных алгоритмов. В таком случае спецификация RTL является результатом работы инструментальных средств и представляет собой совокупность описания цифровых схем и (при необходимости) программного обеспечения. Примерами высокоуровневых языков в данном контексте могут послужить C, XMILE или Lua. Последний



язык используется в системе NITTA, применяемой для реализации и тестирования предлагаемого алгоритма процесса синтеза.

Несмотря на достаточную степень разработанности данной темы [2], существующие средства HLS обладают рядом ограничений. В основном они заключаются в необходимости глубокого понимания схемотехники для эффективного использования, высокой сложности, непрозрачности, специализированности под конкретные виды задач, а также привязанности решений к конкретным поставщикам оборудования [1]. NITTA и другие разрабатываемые САПР специализированных вычислителей являются попыткой решения обозначенных проблем, поэтому усовершенствование процесса синтеза в них с помощью новых подходов является актуальной и практически значимой темой.

Исследователи в данной области сталкиваются с проблемой решения задачи с высокой асимптотической сложностью. HLS включает в себя определение конкретной конфигурации набора используемых вычислительных блоков и шин (регистры, сумматоры, интерфейсы ввода/вывода и прочие), что будет называться далее микроархитектурой. Существует очень много вариантов организации вычислительного процесса для любого заданного на высокоуровневом языке алгоритма, в связи с чем осуществление полного перебора возможных микроархитектур не представляется практически достижимым. Чтобы САПР получала результат за приемлемое время, требуется оптимизировать процесс синтеза с помощью применения решений эвристического характера.

Автору известны решения на основе неполного перебора, суть которых в наложении ограничения на множество рассматриваемых вариантов. Ограничения накладываются из соображений разумности с точки зрения авторов алгоритма. Успешность подобных решений определяется прежде всего удачностью выбранных критериев для формирования множества перебора. Чем меньше мощность этого множества и строже ограничения, тем больше эффективность алгоритма зависит от выбранных критериев, что может

приводить к неоптимальному результату. С другой стороны, ослабление ограничений и увеличение мощности множества перебора неизбежно ведёт к росту вычислительной сложности процесса синтеза. Таким образом, обоснованный выбор критериев ограничения множества перебора является критичным этапом в оптимизации процесса синтеза с помощью алгоритмов неполного перебора.

Цель данного исследования – повысить эффективность процесса синтеза специализированных процессоров за счёт статистически обоснованного эвристического алгоритма неполного перебора с применением подходов из области машинного обучения (МО). Критериями эффективности при этом считаются длительность процесса синтеза и время исполнения управляющей программы.

Основными задачами исследования являются:

- 1) изучение существующих применений МО к аналогичным проблемам;
- 2) формализация задачи из предметной области для решения алгоритмами МО;
- 3) разработка алгоритма процесса синтеза с помощью МО;
- 4) исследование эффективности в соответствии с обозначенными критериями.

# 1 ОБЗОР СУЩЕСТВУЮЩИХ РЕАЛИЗАЦИЙ МЕТОДОВ СИНТЕЗА

Как было отмечено во введении, существуют реализации процесса синтеза, использующие алгоритмы неполного перебора. Их суть заключается в наложении ограничения на множество рассматриваемых вариантов при синтезе.

Перед проектированием собственного решения необходимо сделать обзор существующего опыта в отрасли и проанализировать уже имеющиеся реализации, чтобы, во-первых, понятным образом представить прорабатываемую проблематику, а во-вторых – нагляднее продемонстрировать недостатки существующих решений, попытка исправить которые осуществляется в настоящем исследовании.

## 1.1 Метод синтеза в системе NITTA

В САПР NITTA существует структура данных, представляющая процесс синтеза и принятые решения в виде дерева. Каждая вершина в этом дереве – состояние модели вычислителя, а каждое ребро – переход в соответствии с принятым в процессе синтеза решением. В каждой вершине имеется множество решений, которые можно принять, то есть множество рёбер, куда можно продолжать движение по дереву синтеза.

Пример данного дерева изображен на рисунке 1. Рисунок является снимком диаграммы из пользовательского интерфейса системы NITTA. Текущая вершина выделена цветом. Справа снизу от вершин содержится информация о совершённом при переходе в данную вершину действии. Справа сверху от вершин находится номер вершины в списке действий, доступных в предыдущей вершине. Поддеревья сокращаются с указанием количества скрытых смежных вершин.

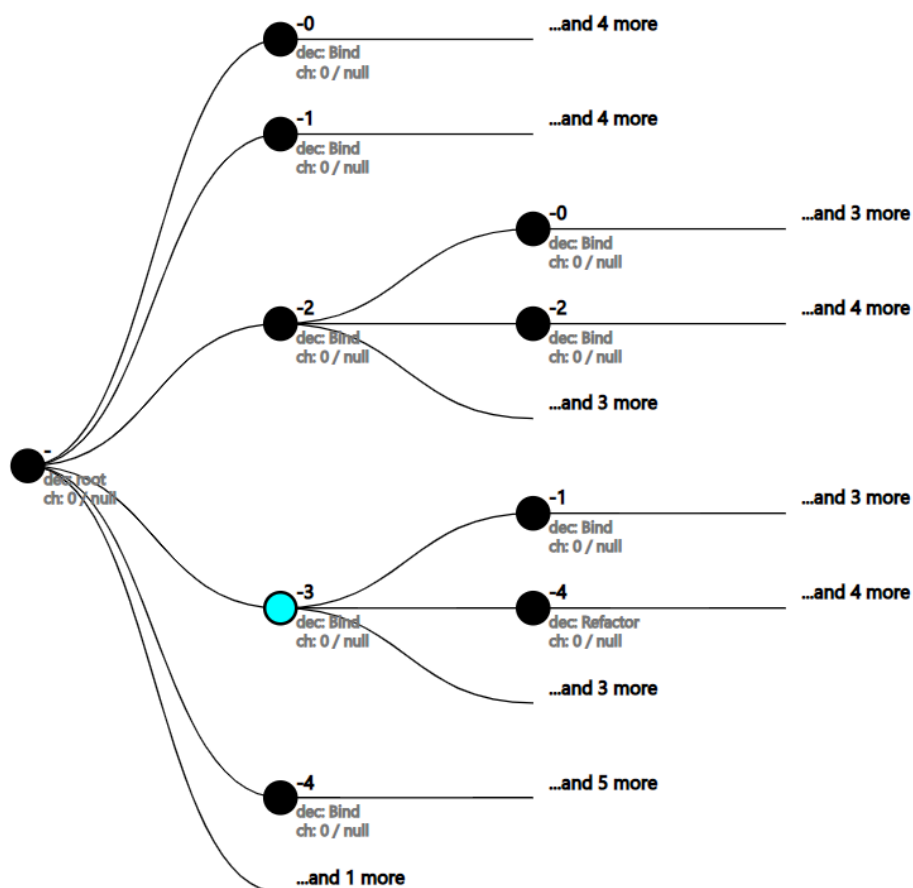


Рисунок 1 – Пример дерева синтеза в САПР NITTA, снимок пользовательского интерфейса

Методом синтеза при данной постановке задачи является способ обхода представленного дерева. Обход производится с корня дерева в глубину посредством посещения вершин-детей. Чем больше вершин посещается, тем больше вычислительных ресурсов необходимо для осуществления процесса синтеза. Отказ от посещения некоторых поддеревьев, таким образом, является способом сэкономить вычислительные ресурсы. Пропуск заведомо невыгодных поддеревьев реализует описанный ранее механизм сокращения множества перебора и, следовательно, ускорения процесса синтеза.

Встаёт проблема выбора порядка обхода дерева, а также проблема определения, стоит ли заходить в поддерево в принципе. Указанные проблемы решаются в системе NITTA локальной оценкой выгоды принятия каждого из возможных решений с помощью действительной целевой функции от выбранных для оценки численных критериев, содержащих в себе информацию

о текущей микроархитектуре и состоянии процесса синтеза. В рассматриваемом примере для выделенной на рисунке 1 вершины доступны действия, представленные на таблицах 1 и 2. Целевая функция обозначена в них как  $Z(d)$ .

Таблица 1 – Пример возможных действий типа «Связывание»

№	Z(d)	Описание	Crit	Lock	Outputs	Alt	Rest	...
0	3052	fram1 – const(1.0) = 1@const#0	0	0	1	2	0	...
1	3042	fram2 – const (1.0) = 1@const#0	0	0	1	2	0	...
2	3597	accum – x1#0 + 1@const#0 = x2#0	0	0	1	1	0	...
3	3610	spi – send(x1#1)	0	0	0	1	0	...

Таблица 2 – Пример возможных действий типа «Переработка»

№	Z(d)	Тип	Описание
4	5000	BreakLoopView	output: x1#0, x1#1 input: x2#0

Перед более подробным описанием содержимого таблиц имеет смысл определить основные типы действий:

- *связывание* (binding decision): назначение конкретному вычислительному блоку (сумматор, регистр, интерфейс) какой-либо конкретной функции (хранение какой-либо константы, вывод определённых данных во внешний интерфейс);
- *переработка* (refactoring decision): преобразование микроархитектуры и/или расписания, преследующее различные цели (например, усовершенствование микроархитектуры тем или иным способом);
- *пересылка данных* (dataflow decision): передача какого-либо значения между различными вычислительными блоками по шине данных.

Разные типы действий имеют различные критерии для оценки их привлекательности. В качестве примера приводятся критерии оценки операции типа «связывание»:

- *critical* (crit): флаг (0 или 1), обозначающий, может ли данное связывание заблокировать другое возможное связывание;

- possibleDeadlock (lock): флаг, обозначающий, может ли связывание вызвать взаимную блокировку;
- outputNumber (outputs): количество выходных переменных;
- alternative (alt): количество альтернативных связываний;
- restless (rest): сколько тактов необходимо для исполнения операции;
- allowDataFlow: сколько пересылок данных может быть исполнено с операцией;
- numberOfBoundFunctions: количество связанных функций;
- percentOfBoundInputs: доля связанных входных переменных.

В таблицах 1 и 2, как и в пользовательском интерфейсе NITTA, представлено, помимо прочего, текстовое описание операций в специальной нотации для отражения их сути. Указывается имя вычислительного блока (например, fram1, fram2, ассум, spi) вместе с описанием связываемой функции, содержащей в себе переменные вместе с индексом их использования в формате «имяПеременной#индексИспользования» (например, x2#0, x1#1). Слева от знака равенства при этом располагается сама функция, справа – выходные переменные.

Алгоритм вычисления целевой функции  $Z(d)$  для каждого типа операций определяется отдельно, но суть текущей реализации везде одинакова: применяется взвешенная сумма критериев или эквивалентное по смыслу выражение. Позитивные веса у критерия повышают вероятность того, что вершине с большим численным значением данного критерия будет отдано предпочтение при обходе дерева. Негативные веса, напротив, понижают эту вероятность.

Веса критериев подобраны вручную авторами системы исходя из соображений разумности. С учётом этого обстоятельства становится очевидным основной недостаток такого подхода, уже упоминавшийся ранее: успешность алгоритма определяется удачностью подобранных весов.

Сбор и анализ каких-либо эмпирических данных позволили бы численно оценить корреляцию между критериями и успешными результатами синтеза,

а также выстроить процесс синтеза таким образом, что потенциально более успешные вершины будут обходиться с гарантированно большей вероятностью, чем потенциально менее успешные. Такой подход повысит степень уверенности в том, что веса критериев в частности и целевая функция в целом подобраны верно.

В связи с этим в рамках настоящего исследования предлагается применить индуктивные алгоритмы машинного обучения для получения статистически обоснованной целевой функции. Такие алгоритмы выявляют общие для какой-либо предметной области закономерности на основе известного конечного набора эмпирических данных. Данная особенность позволяет применять модели МО к проблемам без каких-либо дополнительных знаний о предметной области, основываясь исключительно на математическом анализе эмпирических данных.

В следующем подразделе более подробно рассматриваются параллели между поставленной задачей в рамках реализации процесса HLS и существующими применениями МО к аналогичным проблемам, в частности к поиску выигрышных ходов при реализации игрового искусственного интеллекта посредством анализа графа игровых состояний с помощью применения метода Монте-Карло для поиска в графе.

## **1.2 Применение машинного обучения к деревьям решений**

В результате произведённого обзора исследованных проблем, аналогичных сформулированной задаче поиска наиболее успешных вершин в дереве синтеза, было замечено сходство с задачей, решаемой при реализации игрового искусственного интеллекта для игр, в которых граф игровых состояний может быть сформирован весьма наглядно: шахматы, шашки, Go и другие подобные.

Среди упомянутых игр выделяется Go из-за своей сложности, которая объясняется большим количеством возможных состояний и их динамичностью, что объясняет сложность в реализации превосходящего профессиональных игроков алгоритма относительно шахмат, шашек и некоторых других классических настольных игр с полем [3].

Метод Монте-Карло для поиска в графе (Monte Carlo Tree Search, MCTS) зарекомендовал себя как эффективное средство решения данной задачи [3–5]. Множество рассматриваемых вариантов, аналогично предлагаемой в НИТТА реализации, сокращается посредством статистического анализа успешности конкретных вершин [6]. Когда результат игры становится известным, он распространяется вверх по дереву, меняя информацию на вершинах выше таким образом, чтобы потенциально ведущие к выигрышу вершины выбирались впоследствии чаще. Те вершины, которые выбирались в итоге чаще всего и приводили к наибольшей вероятности побед, и будут становиться следующим выбором по мнению алгоритма. Наглядная и краткая иллюстрация сути алгоритма представлена на рисунке 2.

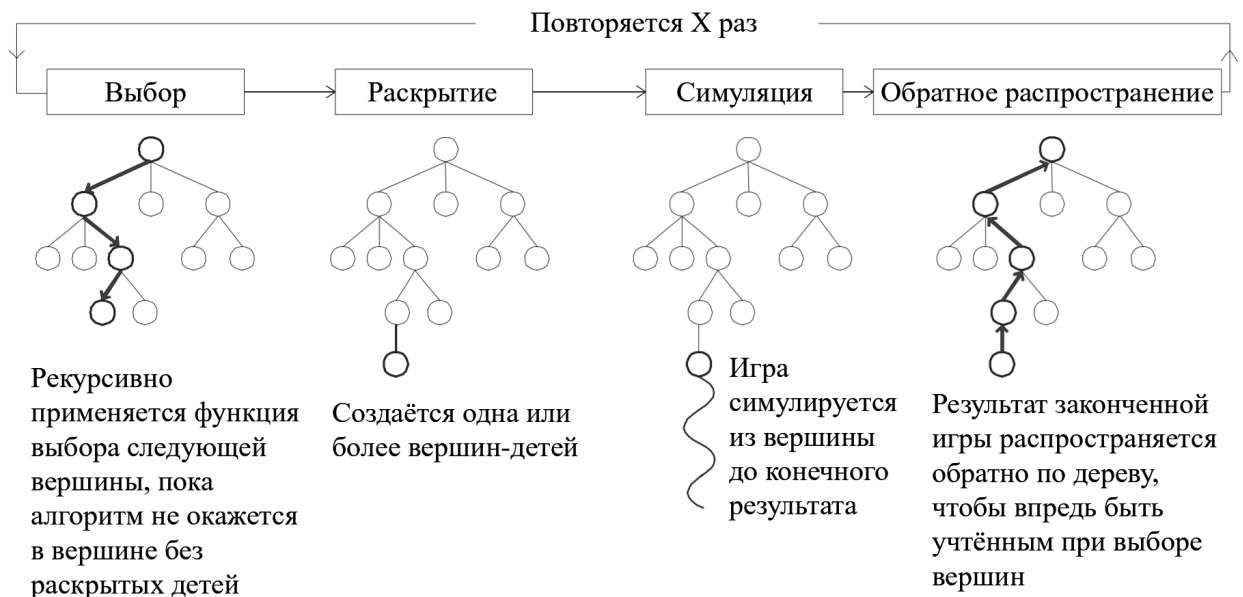


Рисунок 2 – Краткая демонстрация метода MCTS, перевод диаграммы [6]



Исследователями в существенной степени проработана тема применения МО с целью усовершенствования метода MCTS [7, 8]. Некоторые решения используют дополнительные эвристики из предметной области, другие отказываются от них, позволяя алгоритму самостоятельно изучать правила игры и выводить наиболее выигрышные стратегии во время обучения [7].

Рассмотрим, в частности, опыт применения искусственных нейронных сетей на этапе симуляции (в терминологии с рисунка 2), представленном в [7]. Авторами реализуется алгоритм для игры в Dots and Boxes без использования каких-либо знаний из предметной области. Осуществляется попытка заменить их с помощью нейронной сети, предсказывающей вероятность победы. Тренировка осуществляется на данных, сгенерированных самостоятельно в ходе игры системы с самой собой. Сеть используется для оптимизации этапа симуляции: если с точки зрения сети вероятность победы выше порога, то симуляция дальнейших вершин не производится.

Существующий опыт применения метода MCTS вместе с нейронными сетями к задачам из неигровых областей (комбинаторная оптимизация, удовлетворение ограничений, задачи планирования, процедурной генерации контента и др.) даёт дополнительные основания полагать, что некоторые идеи данного метода могут быть успешно применимы к задаче из области реализации HLS в САПР специализированных процессоров [7].

Решаемая задача, тем не менее, имеет свою специфику. В частности, нет необходимости в каждой конкретной вершине давать конечный, необратимый результат – выбранный алгоритмом ход, который требуется в играх. Эта особенность мотивирует не применять метод MCTS, заточенный под данное ограничение, как есть, а использовать лишь определённые подходы из упомянутых вариаций метода: самостоятельную генерацию тренировочных данных в виде заранее просчитываемых деревьев, использование нейронных сетей в качестве функции оценки при продвижении по дереву, а также саму суть метода Монте-Карло – использовать вероятностный подход, основанный

на статистических аппроксимациях, для решения детерминированных задач, которые, однако, слишком вычислительно сложны для решения детерминированными алгоритмами.

### **1.3 Выводы**

Произведён обзор существующих реализаций метода синтеза. Детальнее продемонстрирована решаемая проблема, обозначены недостатки существующих решений. Кроме того, исследованы применения машинного обучения к аналогичным проблемам. Показана применимость некоторых идей из других областей, в которых решались похожие задачи, к проблеме из области реализации HLS в САПР специализированных процессоров.

## **2 ПРОЕКТИРОВАНИЕ РЕШЕНИЯ НА БАЗЕ МАШИННОГО ОБУЧЕНИЯ**

В дальнейших разделах описывается, реализуется и тестируется решение, представляющее собой попытку исправить обозначенные недостатки реализации HLS в САПР специализированных процессоров NITTA. Предлагаемое решение основано на рассмотренных в предыдущем разделе идеях.

### **2.1 Формализация задачи для алгоритмов машинного обучения**

Как было описано ранее, предлагается применить алгоритмы МО для подбора статистически обоснованной целевой функции, которая будет представлена моделью МО. Сама модель будет подробнее рассмотрена в следующем подразделе. Предлагается сначала сконцентрироваться на преобразовании эмпирических данных из системы для использования в качестве входных данных модели, проектировании меток тренировочных данных, к которым будут стремиться ответы модели, а также описании роли модели в процессе синтеза.

На вход модель принимает упомянутые ранее критерии с информацией о возможном действии в дереве синтеза. В качестве модели предлагается использовать искусственную нейронную сеть, поэтому все входные данные должны быть преобразованы к численному виду. Произведён анализ типов данных критериев оценки различных видов операций, доступных в процессе синтеза. Определены необходимые преобразования.

Кроме критериев оценки для различных видов операций, во входные данные добавлена важная информация о текущей вершине, чтобы модель могла её учитывать в процессе тренировки. Включение вида рассматриваемой операции может помочь модели быстрее сходиться и демонстрировать более

точные результаты в условиях, когда имеется необходимость проставлять искусственные значения (например, -100) вместо отсутствующих значений критериев для операций других видов. Такая необходимость возникает в связи с тем, что размерность вектора входных данных и смысл чисел в нём должен быть единым для всего набора, содержащего три различных вида операций с разным набором критериев, при этом каждая конкретная строка в наборе данных обозначает операцию лишь одного вида. Включение количеств различных доступных операций на текущей вершине может помочь учитывать контекст оценки и соблюдать баланс между разными типами операций.

Входные данные, таким образом, имеют следующую структуру:

- 1) блок критериев операций типа «связывание» (их смысл описан ранее):
  - а) `critical`: флаг (0 или 1), допустимо использование как есть;
  - б) `possibleDeadlock`: флаг, допустимо использование как есть;
  - в) `outputNumber`: число, используется как есть;
  - г) `alternative`: число, используется как есть;
  - д) `restless`: число, используется как есть;
  - е) `allowDataFlow`: число, используется как есть;
  - ж) `numberOfBoundFunctions`: число, используется как есть;
  - з) `percentOfBoundInputs`: число, используется как есть.
- 2) блок критериев операций типа «переработка»:
  - а) `type`: категориальный тип, требуется преобразование;
- 3) блок критериев операций типа «пересылка данных»:
  - а) `waitTime`: число, используется как есть;
  - б) `restrictedTime`: флаг, допустимо использование как есть;
  - в) `notTransferableInputs`: массив чисел, по смыслу допустимо использовать сумму;
- 4) дополнительная информация:
  - а) вид операции: категориальный тип, требуется преобразование;

- б) количество доступных операций вида «связывание»: число, используется как есть;
- в) количество доступных операций вида «переработка»: число, используется как есть;
- г) количество доступных операций вида «пересылка данных»: число, используется как есть;

Во входных данных есть поля категориального (номинального) типа, значения которых качественно характеризуют описываемый объект, относят его к какой-либо номинальной категории и не имеют отношения порядка. Использовать их как есть с нейронными сетями нельзя, необходимо преобразование к численному вектору.

Самым популярным из относительно эффективно решающих эту задачу преобразованием является кодирование с использованием индивидуальных переменных для каждой категории (One-Hot Encoding, ONE) [9]. Например, для преобразования категориального типа цвета, значения которого задаются тремя категориями (красный, зелёный, синий), будет использоваться векторы  $\bar{x} \in \mathbb{R}^3$ :

- красный – (1, 0, 0);
- зелёный – (0, 1, 0);
- синий – (0, 0, 1).

Таким образом, применяя ONE, станет возможно использование категориальных данных как входных для нейронной сети. Одно поле категориального типа с мощностью  $n$  при этом будет заменено вектором  $\bar{x} \in \mathbb{R}^n$ .

Далее рассматривается способ вычисления меток для тренировочных данных, к которым будут стремиться ответы модели. Тренировочные данные будут генерироваться в виде относительно полных заранее вычисляемых деревьев синтеза для различных примеров высокоуровневых алгоритмов.

Вычислительная сложность процесса генерации таких деревьев хоть и высока, но позволяет это сделать один раз за приемлемое время.

Листьями в сгенерированных деревьях будут различные микроархитектуры и расписания программ – результаты синтеза. Каждый лист возможно оценить с помощью метрик, чтобы учитывать качество результатов синтеза при формировании меток. В настоящем исследовании из метрик, относящихся к качеству результатов, минимизируется длительность работы конечного алгоритма, но такой же подход позволяет оптимизировать и другие метрики: количество используемых вычислительных блоков, объём используемой памяти, любые другие численные показатели.

Кроме того, минимизируется количество вершин, которые необходимо пройти для достижения результата заданного качества, но это имеет меньший вес при вычислении финальной метки, так как, хоть качественный результат и является приоритетом, при прочих равных желательно достичь его в процессе синтеза как можно быстрее.

Ещё одним важным обстоятельством является то, что при тренировке модели на данных, собранных с различных высокоуровневых алгоритмов, имеет смысл только относительная длительность процесса, а не абсолютная. Например, результаты синтеза для алгоритма А будут в среднем предлагать решения в 7 тактов, а для алгоритма Б в 152 такта, так как алгоритм Б сложнее алгоритма А. Чтобы у более сложных алгоритмов не было большего веса в метках по исключительно численным причинам, предлагается нормализовать метрики, сведя их стандартизированную оценку (z-score) к 1 следующим образом:

$$x_{\text{норм}} = \frac{x - \bar{x}}{\sigma}, \quad (1)$$

где  $x_{\text{норм}}$  – нормализованная метрика,  $x$  – исходная метрика,  $\bar{x}$  – среднее значение величин метрики,  $\sigma$  – стандартное отклонение величин метрики. Все нормализованные таким образом метрики будут иметь среднее значение 0, а

их стандартное отклонение будет равно 1. Конечная метка при этом будет вычисляться как взвешенная сумма нормализованных метрик.

Сбор тренировочных данных предлагается производить двухкратным обходом деревьев синтеза в глубину. Необходимо два обхода, так как невозможно начать нормализовывать метрики, не имея данных о распределении величин метрик со всех листьев дерева.

В первый обход производится непосредственный расчёт моделей вычислителей и сбор величин метрик в листьях для расчёта параметров их распределений. Во второй обход – формирование набора данных. Каждая запись соответствует ребру, то есть переходу из вершины-родителя в вершину-ребёнка. В каждой записи – входные данные обозначенного формата, а также метка.

Если вершина-ребёнок – лист, то метка вычисляется непосредственно как взвешенная сумма нормализованных метрик в случае успешного синтеза. В случае неуспешного синтеза меткой считается искусственное отдельно задаваемое значение, соответствующее плохой оценке, чтобы сохранить отрицательные примеры в наборе данных.

Когда у вершины есть дети, появляется необходимость агрегировать метрики поддеревьев. Встаёт вопрос, ответ на который дать на этапе проектирования затруднительно: что важнее – средние значения метрик листьев поддерева или один лишь максимум, отражающий наилучший результат, существующий в этом поддереве?

Чтобы поставить вопрос нагляднее, рассмотрим пример на рисунке 3. Предположим, что нам нужно рассчитать две метки для рёбер из корня (вершина «К»), то есть оценить два поддерева: А и Б. На листьях проставлена условная метка, являющаяся численной оценкой удачности результата, больше – лучше. У поддерева А стабильно более высокая оценка листьев, максимум – 16. У поддерева Б, тем временем, средняя оценка листьев гораздо ниже, но из-за особенностей конкретного алгоритма есть один удачный лист с оценкой в 17 (выделен серым), что делает максимум поддерева Б равным 17.

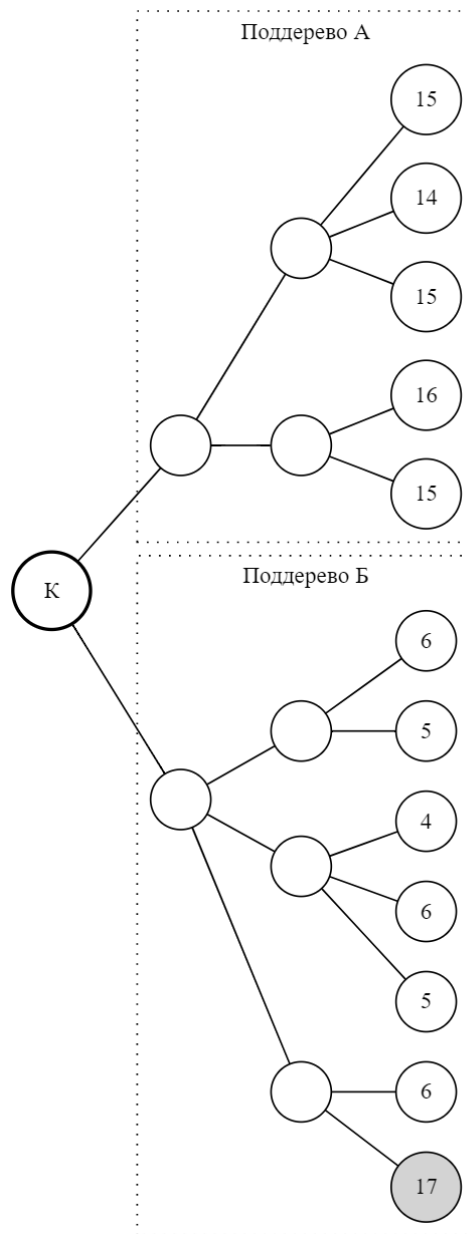


Рисунок 3 – Пример-иллюстрация к вопросу о расчёте меток поддеревьев

Это больше, чем у поддерева А, хотя обычно решение выбирать действие, ведущее к поддереву А, было бы более разумным.

На первый взгляд может показаться, что, так как конечный результат берётся в единственном экземпляре, важен лишь максимум, но из этого следует, что какой-либо максимум, который может быть единичным выбросом и частным случаем, будет являться и меткой для решений у самого корня дерева, что, вероятно, будет лишь «запутывать» модель. Если есть «хорошее» или «плохое» решение непосредственно у корня, то оно будет приводить к



«хорошим» или «плохим» средним метрикам во всём поддереве. В противном же случае это может оказаться выбросами и лишней информацией, которая будет мешать эффективной тренировке.

Таким образом, если при поиске ответа на этот вопрос задуматься над тем, какие метки должны быть у больших поддеревьев, а именно – какой «правильный» ответ можно поставить на вопрос об удачности действий, приводящих в эти самые большие поддеревья, то роль средних величин метрик в поддереве становится нагляднее.

Было решено выбрать гибкий подход и ввести параметр  $\lambda \in [0; 1]$  – вес максимально успешного листа в метке поддерева. Таким образом, расчёт метки поддерева будет производиться из двух компонент (наилучший результат и средний результат) следующим образом:

$$z = \lambda * y_{max} + (1 - \lambda) * y_{avg} , \quad (2)$$

где  $z$  – вычисляемая метка поддерева,  $y_{max}$  – максимальная метка листа поддерева,  $y_{avg}$  – средняя величина меток листьев поддерева. В рамках данного исследования берётся  $\lambda = 0,6$ .

Таким образом, при втором обходе дерева для всех рёбер описанным способом создаются записи для последующей тренировки модели со входными данными и соответствующими метками. Метки при этом обладают требуемыми свойствами: независимость от синтезируемого алгоритма и пропорциональность вероятности нахождения относительно успешного решения при принятии соответствующего метке решения. Ответы натренированной модели будут стремиться к этим меткам, следовательно, эти ответы могут быть использованы непосредственно как значения целевой функции  $Z(d)$ , что определяет роль модели в процессе синтеза.

## 2.2 Описание модели

Прежде всего, стоит отметить, что модель, используемая в рамках данной работы, предлагается как базовая. Её цель – показать применимость подхода в принципе, поэтому приемлемого результата будет достаточно. Его улучшение должно стать предметом дальнейших исследований и оставлено за пределами текущей работы в связи с большим объёмом.

Предлагается применить относительно простую нейронную сеть, похожую по своей сути на сеть из уже упомянутой статьи [7]. Схематическое изображение представлено на рисунке 4. Количество нейронов в слоях на диаграмме из соображений наглядности сокращено, действительные значения отражены в подписях. Толщина связей между нейронами пропорциональна условным весам.

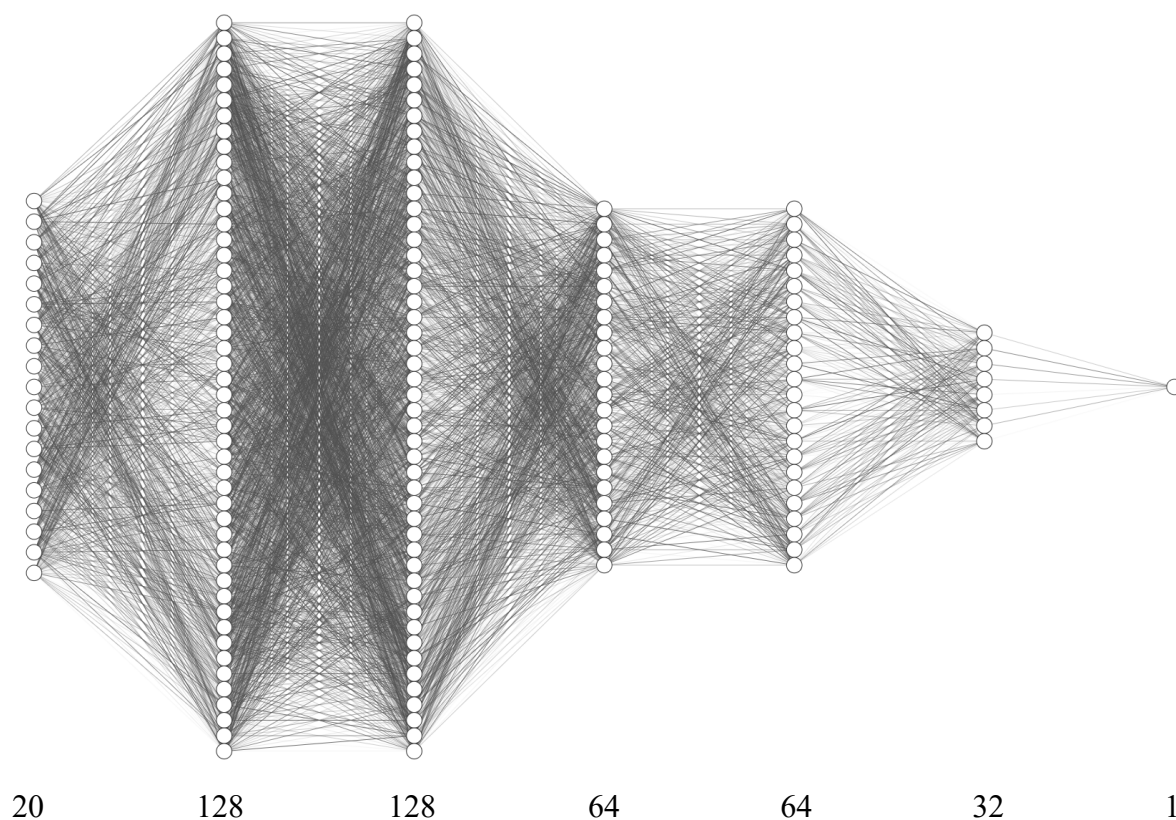


Рисунок 4 – Схематическое изображение предлагаемой нейронной сети

На вход подаётся, в соответствии с предыдущим подразделом, вектор из 20 чисел (преобразование ONE уже применено), на выходе – 1 число. Все слои полносвязные. Ко всем слоям, кроме первого и последнего, применяется активационная функция – линейная ректификация (Rectified Linear Unit, ReLU). Она необходима в качестве нелинейного преобразования, делающего нейронную сеть нелинейной функцией, которая способна успешно повторять нелинейные зависимости между входными и выходными данными.

Кроме того, ко всем скрытым слоям, кроме последнего, применяется L2-регуляризация, что помогает бороться с переобучением и получать более стабильный тренировочный процесс при таком же размере сети и коэффициенте скорости обучения (learning rate).

Для тренировки предлагается использовать оптимизатор Adam [10]. Он обладает рядом преимуществ перед стохастическим градиентным спуском и другими классическими методами оптимизации, в частности способностью адаптировать коэффициент скорости обучения, лучшей сходимостью и простотой конфигурации.

В качестве минимизируемой функции ошибки предлагается использовать среднеквадратическую ошибку (Mean Squared Error, MSE). Так как применяется регуляризация, влияющая на функцию ошибки, имеет смысл производить расчёт сторонней метрики для оценки модели в процессе тренировки. Предлагается средняя абсолютная величина ошибки (Mean Absolute Error, MAE) как один из вариантов, имеющих наиболее наглядный физический смысл.

### **2.3 Выводы**

Таким образом, определён способ применения алгоритмов МО к решаемой проблеме, а именно формат и необходимые преобразования данных из системы NITTA к подаче на вход нейронной сети. Спроектированы метки

и процесс сбора тренировочных данных, к которым будут стремиться ответы модели, а также описана роль модели в процессе синтеза. Кроме того, спроектирована сама нейронная сеть, определены необходимые параметры тренировочного процесса. Имеется готовый проект предлагаемого решения. Следующими этапами будет его реализация и оценка.

## **3 РЕАЛИЗАЦИЯ РЕШЕНИЯ НА БАЗЕ МАШИННОГО ОБУЧЕНИЯ**

Для программной реализации предложенных алгоритмов выбран язык Python 3.7 в связи с развитой инфраструктурой для решения задач МО и интерпретируемой природой, удобной для написания алгоритмов анализа данных [11]. Используются библиотеки Tensorflow [12] и Pandas [13], а также другие: NumPy, asyncio + aiohttp, dataclasses\_json. В качестве интерактивной среды использовался Jupyter Notebook.

### **3.1 Сбор тренировочных данных**

Прежде всего необходимо организовать сбор набора данных для обучения модели. Для этого потребуется обращаться к системе NITTA и извлекать данные о деревьях синтеза для различных примеров. Общение реализовано с помощью имеющегося в системе NITTA веб-интерфейса. Полный исходный код реализации сборщика данных приведён в приложении А.

САПР запускается для различных входных алгоритмов и занимает определённый порт, по которому будет происходить коммуникация. Используются следующие методы:

- GET /node/{node\_id} – возвращает информацию о вершине;
- GET /node/{node\_id}/subForest – возвращает информацию о детях вершины.

Идентификаторы вершин имеют вид разделённых дефисами порядковых номеров в списках возможных действий их вершин-родителей. Например, «-0-0-4-1» – вершина глубины 4. Корень дерева имеет идентификатор «-». В коде идентификаторы вершин обозначаются как «sid» в соответствии со сложившимися при разработке САПР NITTA соглашениями.

В коде, кроме того, под «subtree» понимается поддерево текущей вершины, включающее её саму, что важно не путать с «subforest», обозначающим исключительно массив поддеревьев-детей.

Реализована структура данных NittaNode, в которую производится разбор получаемых от САПР ответов. Кроме того, для этой структуры данных реализованы все вычисления и алгоритмы, потребовавшиеся в работе далее. Основные поля и операции класса изображены на рисунке 5.

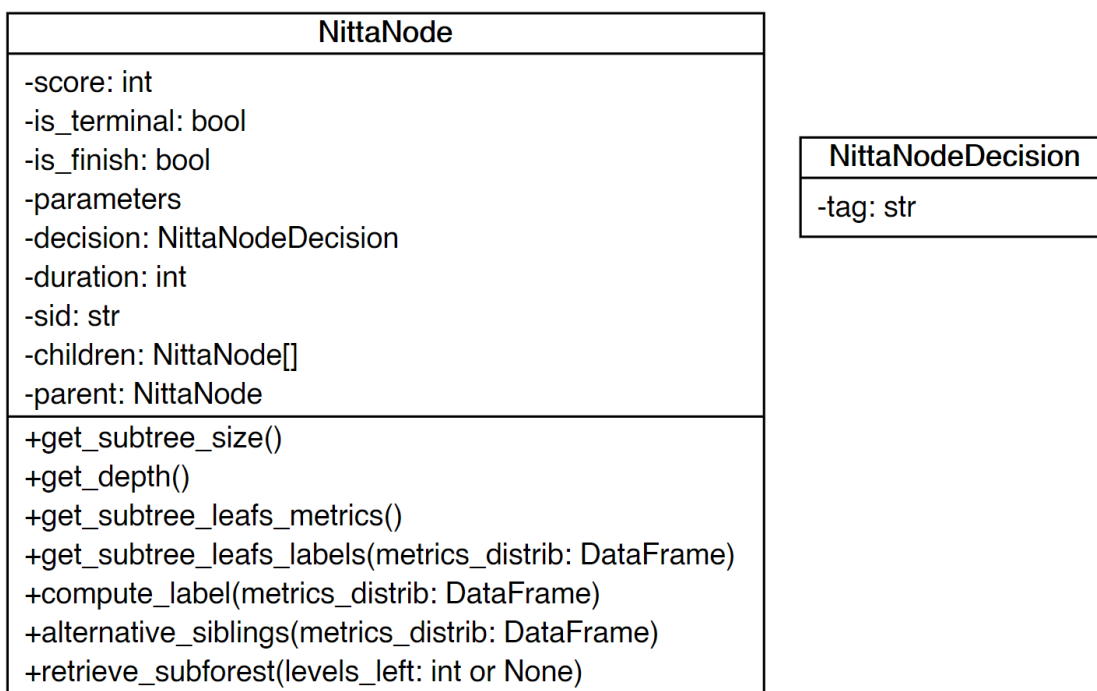


Рисунок 5 – Диаграмма класса NittaNode и связанных классов с основными полями и операциями

Поля имеют именно такую структуру в первую очередь потому, что происходит автоматический разбор в данный тип получаемого от системы NITTA текстового представления вершин в формате JSON. Разбор реализован с помощью декоратора «@dataclass\_json» из библиотеки dataclasses-json. Поля children и parent добавлены вручную для обеспечения связности дерева и возможности навигации по нему.

Операции реализуют необходимые в дальнейшем вычисления, которые необходимо совершать с полученными данными. Так как осуществляется работа с деревом, почти все операции удобно было реализовать рекурсивно.

Некоторые операции исполнены в виде свойств, некоторые – в виде явных функций.

Смысл реализованных операций заключается в следующем.

- Свойство `subtree_size` вычисляет размер поддерева вершины, включая её саму.
- Свойство `depth` вычисляет глубину текущей вершины в дереве. Глубиной корневой вершины считается 0.
- Свойство `subtree_leafs_metrics` возвращает структуру данных с имеющимися в поддерева метриками листьев для последующего анализа распределения их величин.
- Функция `get_subtree_leafs_labels` принимает данные о распределении величин метрик листьев во всём дереве (получается операцией `subtree_leafs_metrics` от корневой вершины) и возвращает массив имеющихся меток листьев для текущего поддерева. Используется для вычисления меток корней поддерева.
- Функция `compute_label` принимает данные о распределении величин метрик листьев во всём дереве и возвращает вычисленную для текущей вершины тренировочную метку.
- Свойство `alternative_siblings` возвращает информацию о количестве альтернативных решений, возможных из вершины-родителя.
- Процедура `retrieve_subforest` осуществляет заполнение массива `children` информацией о детях, полученной от системы NITTA по сети. Используется при построении дерева. Опционально можно передать максимальную глубину проработки (0 – только свои дети, 1 – свои дети и дети детей, и так далее).

Кроме того, реализована процедура `retrieve_whole_nitta_tree`, позволяющая получить полное дерево синтеза. Демонстрация её работы приведена на рисунке 6.

```
In [141]: ▶ nitta_tree = await retrieve_whole_nitta_tree()
print(f"Nodes: {nitta_tree.subtree_size}")
nitta_tree

Finished tree retrieval in 6.55 s
Nodes: 5003

Out[141]: NittaNnode(score=0, is_terminal=False, is_finish=False, parameters='root',
decision=NittaNnodeDecision(tag='RootView'), duration=0, sid='-')
```

Рисунок 6 – Демонстрация работы процедуры выгрузки дерева

Пример обработки данных дерева с использованием реализованной структуры изображён на рисунке 7.

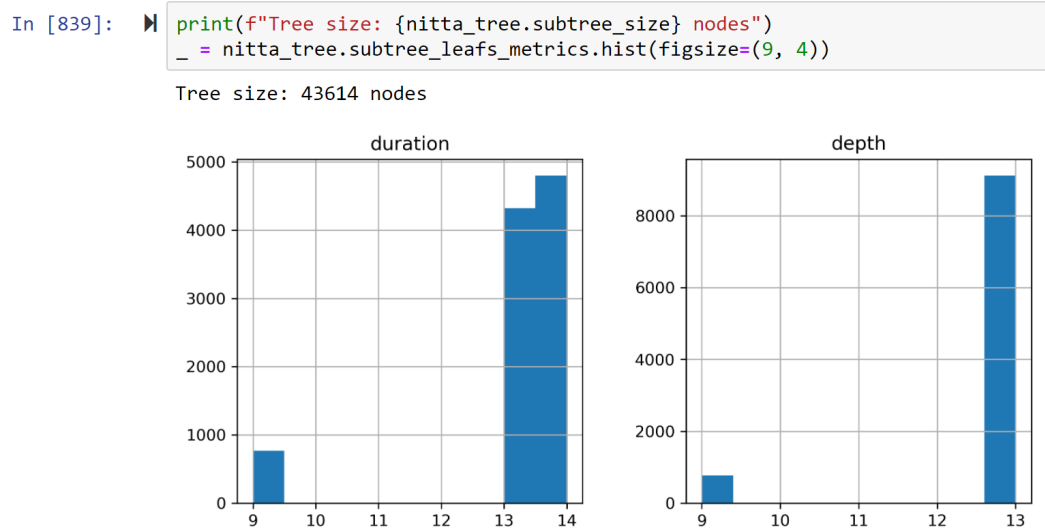


Рисунок 7 – Пример использования реализованной структуры данных для получения распределений метрик листьев дерева

Получившиеся гистограммы характеризуют распределения метрик листьев-результатов во взятом в качестве примера дереве на 43614 вершин. Метриками, как описано ранее, является длительность получившегося расписания в тактах (duration) и глубина листа (depth).

В табличный формат полученное дерево преобразуется с помощью реализованной функции `assemble_tree_dataframe`. Для каждого ребра рекурсивно формируется запись, содержащая всю необходимую информацию для формирования набора входных данных. При движении вверх по стеку вызовов результаты объединяются в одну таблицу. Пример использования изображён на рисунке 8.



```
In [861]: ▶ tdf = assemble_tree_dataframe("spi3", nitta_tree)
display(tdf)
```

tag	old_score	is_leaf	alt_bindings	alt_refactorings	alt_dataflows	pOutputNumber	pAlternative	...
BindDecisionView	3410.0	False	4	1	0	0.0	1.0	...
BindDecisionView	3472.0	False	3	1	0	1.0	1.0	...
BindDecisionView	3522.0	False	2	0	0	1.0	1.0	...
BindDecisionView	3542.0	False	1	0	0	1.0	1.0	...
BindDecisionView	3534.0	False	0	0	1	2.0	1.0	...
...	...	...	...	...	...	...	...	...

Рисунок 8 – Пример использования функции `assemble_tree_dataframe`

Также реализована процедура `process_example`, автоматизирующая запуск САПР для различных примеров, обработку дерева и формирование конечного табличного набора данных, аналогичного по структуре выводу функции `assemble_tree_dataframe`, в формате CSV.

С помощью данной программы были получены подобные таблицы для многих примеров высокоуровневых алгоритмов из репозитория САПР NITTA. Далее эти таблицы будут использоваться в качестве тренировочных данных.

### 3.2 Подготовка данных

Как было отмечено ранее, некоторые поля во входных данных требуют преобразования. Кроме того, для тренировочного процесса необходимо обеспечить объединение данных в группы (`batching`), перемешивание и повторение набора, если данные кончились. Также предлагается оставить 20% имеющихся тренировочных данных на валидацию модели.

Таким образом, полученные в предыдущем разделе данные требуют дальнейшей обработки перед тренировкой. Подготовка данных, реализация и

обучение модели выполняется второй программой, исходный код которой находится в приложении Б.

Подготовка данных осуществляется в рамках функций `preprocess_df` и `create_datasets`. Первая функция принимает на вход таблицу, полученную объединением CSV-файлов для различных высокоуровневых алгоритмов. Данные файлы были получены ранее как результат работы программы из приложения А. Функция преобразует флаги в числа, раскрывает категориальные типы в численные векторы описанным способом (ONE), удаляет лишние столбцы, а также заполняет отсутствующие данные искусственным значением – 0.

Пример результата работы функции `preprocess_df` представлен в таблице 3. Этот пример является используемым тренировочным набором данных. В нём 245816 строк и 21 колонка.

Таблица 3 – Пример результата работы функции `preprocess_df`

	<b>label</b>	<b>alt_bindings</b>	<b>alt_refactorings</b>	<b>alt_dataflows</b>	<b>pOutputNumber</b>	<b>...</b>
<b>0</b>	0.656986	5	0	0	1.0	...
<b>1</b>	-0.274288	3	0	0	2.0	...
<b>2</b>	-0.169982	1	1	0	1.0	...
<b>3</b>	-0.071237	0	1	1	0.0	...
<b>4</b>	-0.071237	0	1	0	0.0	...
<b>...</b>	...	...	...	...	...	...
<b>245811</b>	3.548563	0	0	0	0.0	...
<b>245812</b>	3.548563	0	0	0	0.0	...
<b>245813</b>	3.548563	0	0	1	0.0	...
<b>245814</b>	3.548563	0	0	0	0.0	...
<b>245815</b>	3.548563	0	0	0	0.0	...

Функция `create_datasets`, в свою очередь, занимается разделением указанного набора на тренировочный и тестовый (196652 и 49164 строк соответственно), разделением на входные и выходные данные, а также их группировкой, повторением и перемешиванием с помощью средств модуля `tensorflow.data`. Подобранный размер группы для тренировочного набора данных – 16 строк.

Полученные объекты (`train_ds`, `test_ds`) могут быть переданы непосредственно модели.

### 3.3 Реализация, обучение и тестирование модели

В программе, исходный код которой представлен в приложении Б, также реализована сама модель, процесс её тренировки и сохранение на диск. Создание модели выполняется функцией `create_model`. Используется интерфейс Keras, поэтому все описанные в предыдущей главе концепты применяются декларативно и могут быть гибко изменены при экспериментировании с архитектурой модели.

Созданная модель была визуализирована средствами Keras (рисунок 9). Она имеет 33729 тренируемых параметров. На диаграмме также указаны размеры входных (`input`) и выходных (`output`) векторов на каждом слое. Знак вопроса означает размер группы тренировочных записей, который является внешним по отношению к модели параметром набора данных, поэтому тут не указывается.

В конце приложения Б приводится код, занимающийся непосредственно тренировкой и сохранением модели. Сохранённая на диск модель может впоследствии эксплуатироваться в качестве целевой функции в процессе синтеза без каких-либо дополнительных зависимостей от данных или реализации модели.

Пример визуализации тренировочного процесса представлен на рисунке 10, где по оси абсцисс идут эпохи тренировки, а по оси ординат – параметры модели.

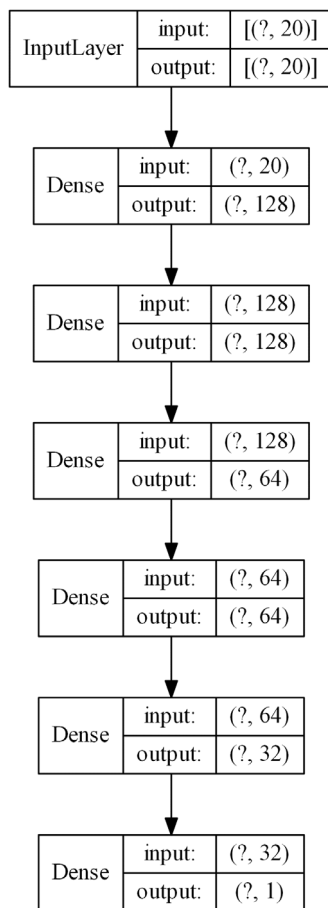


Рисунок 9 – Визуализация реализованной модели

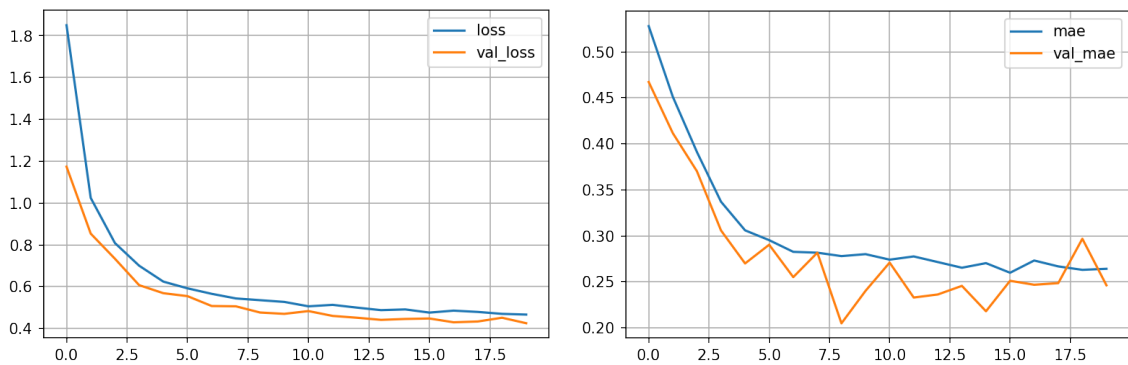


Рисунок 10 – Визуализация тренировочного процесса

Ошибка (*loss*) с течением времени убывает как на тренировочных, так и на тестовых данных (*val\_loss*). MAE при этом отличается более заметными на графиках флуктуациями, что вкупе со стремлением скорости уменьшения ошибки к нулю обосновывает остановку тренировочного процесса в изображённом состоянии.

### 3.4 Выводы

Выбранный стек технологий позволил эффективно реализовать спроектированное решение. Для решения значительной части задач удалось переиспользовать код библиотек общего назначения, не привязанных к предметной области. Реализованные классы показали себя удобными для достижения поставленных целей, алгоритмы анализа данных были написаны лаконично и быстро, а среда разработки и интерпретируемость языка Python позволили работать с реализацией и данными в исследовательском стиле.

Однако, у выбранного подхода есть свои проблемы, например, относительно долгий сбор данных по сети. Есть примеры относительно сложных высокоуровневых алгоритмов, полное дерево которых с текущей скоростью будет обходиться несколько лет. Процесс можно значительно ускорить, если локализовать формирование CSV-таблиц указанного формата непосредственно внутри САПР NITTA. Это может стать направлением дальнейших исследований.

Реализованная модель продемонстрировала сходимость процесса тренировки, что говорит о практической возможности решения поставленной задачи представленным методом. Тем не менее, модель, как было обозначено ранее, имеет значительный потенциал для улучшения. Необходимы, в частности, поиск способов получения меньших значений MAE натренированной модели, а также попытки применить другие классы моделей МО к решаемой задаче (например, деревья принятия решений), что также может стать предметом дальнейших исследований.

## 4 ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ РЕШЕНИЯ

### 4.1 Определение метода

Для исследования эффективности реализованного решения и его сравнения с существующим алгоритмом вычисления целевой функции в САПР NITTA предлагается поставить их в равные условия и сопоставить полученные двумя алгоритмами результаты в соответствии с обозначенными ранее критериями эффективности: длительностью процесса синтеза и временем исполнения управляющей программы.

Длительность процесса синтеза при этом считается пропорциональной количеству пройденных вершин. В самом простом случае допустимо пренебречь стоимостью вычисления непосредственно целевой функции и сконцентрироваться на количестве пройденных вершин, так как вычислительная стоимость расчёта каждой новой непосещённой вершины считается значительно превосходящей стоимость вычисления целевой функции. В настоящем исследовании при оценке результатов, тем не менее, учитывается количество вызовов целевой функции.

В качестве простого метода синтеза предлагается численно оценивать целевой функцией всех детей текущей вершины, а затем спускаться в  $n \in \{1, 2, \dots\}$  наилучших. Подходит минимальное  $n$ , при котором решения существуют для обоих алгоритмов. Рекурсивный спуск в детей повторяется до тех пор, пока алгоритм не окажется в листе. Если лист соответствует успешному синтезу, он возвращается как лучший по мнению алгоритма и подлежит оценке в соответствии с обозначенными критериями. Если лист не соответствует успешному синтезу, обход продолжается по следующим детям в установленном порядке. При таком подходе различаться будет только непосредственно метод вычисления целевой функции, что отвечает

поставленному требованию о равных условиях. Сравнение методов должно производиться для различных примеров высокоуровневых алгоритмов.

## 4.2 Проведение исследования

Относительно простым способом организовать равные условия для целевых функций, реализованных на разных языках, будет использование существующего веб-интерфейса и сериализованных оценок.

Предлагается использовать существующий класс `NittaNode`, способный работать с веб-интерфейсом САПР NITTA и реализующий высокоуровневый интерфейс к операциям над деревом синтеза и данными в нём. Старые значения целевых функций получаются через веб-интерфейс и сохраняются в поле `score`. Новые значения могут быть рассчитаны с использованием существующих операций над данным классом.

Реализация описанного метода синтеза представлена в листинге 1. Используется концепция численного оценщика вершин (`evaluator`), являющегося функцией от вершины. Алгоритм рекурсивно вызывается от детей, имеющих наибольшую величину от оценщика.

Листинг 1 – Реализация простого метода синтеза

```
async def select_best_by_evaluator(session, evaluator, node, children_limit=None):
    if node.is_leaf:
        return node if node.is_finish else None

    await node.retrieve_subforest(session, 0)
    children = [(evaluator(child), child) for child in node.children]
    children.sort(key=lambda v: v[0], reverse=True)
    if children_limit:
        children = children[:children_limit]

    while children:
        next_best_child = children.pop(0)[1]
        result = await select_best_by_evaluator(session, evaluator, next_best_child,
        children_limit)
        if result is not None:
            return result

    return None
```

Для проведения исследования необходимо реализовать два оценщика: для нового и старого методов вычисления целевой функции. Оценщик для старого метода будет коротким, так как мы уже получили необходимую оценку по сети. Он представлен на листинге 2.

#### Листинг 2 – Реализация оценщика вершин старого метода

```
def old_evaluator(node: NittaNode) -> int:  
    return node.score
```

Оценщик для нового метода переиспользует существующие функции для подготовки данных вершины. Представленные на листинге 3 манипуляции связаны с обеспечением правильного порядка чисел в векторе, подаваемом на модель. С этой целью имеется локальный список колонок (`final_columns`). Модель для оценки загружается с диска. Сформированный из вершины числовой вектор подаётся на модель, а её ответ возвращается в качестве значения целевой функции.

#### Листинг 3 – Реализация оценщика вершин нового метода

```
MODELS_ROOT = Path("models")  
model = tf.keras.models.load_model(MODELS_ROOT / "model1")  
  
def new_evaluator(node: NittaNode) -> float:  
    node_df = assemble_tree_dataframe("", node, include_label=False, levels_left=-1)  
    filled_metrics_df = pd.concat([pd.DataFrame(columns=metrics_columns), node_df])  
    preprocessed_df = preprocess_df(filled_metrics_df)  
    right_final_columns_df = pd.concat([pd.DataFrame(columns=final_columns),  
                                       preprocessed_df])[final_columns]  
    ohe_flags_zero_filled_df = right_final_columns_df.fillna(0)  
    final_df = ohe_flags_zero_filled_df  
  
    return model.predict(final_df.values)[0][0]
```

Полный исходный код реализации программы для испытаний эффективности приведён в приложении В.

Программа была применена к нескольким примерам. Некоторые из них имели сложные деревья синтеза с большим числом вершин, некоторые были относительно простыми. Кроме того, эксперименты «constantFolding» и



«fibonassi» проводились на примерах алгоритмов, небольшая часть деревьев синтеза которых присутствовала в тренировочных данных. Они приведены для справки и оценивались для контроля на предмет переобученности и «здравомыслия» модели (sanity check). Параметры результатов синтеза с различными оценщиками приведены в таблице 4.

Таблица 4 – Результаты сравнения эффективности старого и нового алгоритмов вычисления целевой функции

Эксперимент	Длительность результата, такты		Глубина листа		Вызовов оценщика	
	Старый	Новый	Старый	Новый	Старый	Новый
constantFolding	10	11	13	15	1635	16
fibonacci	6	6	9	8	10	9
spi1*	5	5	5	5	6	6
spi2*	7	7	7	7	8	8
shift*	6	6	5	5	6	6
doubleReceive*	7	7	6	6	7	7
teacup	26	21	27	27	28	1367

В экспериментах, отмеченных «\*», оба алгоритма дали наилучший возможный результат. Количество вызовов оценщика в некоторых экспериментах бывает значительно выше среднего как у новой целевой функции, так и у старой, и связано это прежде всего с предложенной реализацией простого метода синтеза, а именно с обходом лучших детей в глубину. Целевая функция в связи с локальностью оценки иногда может приводить процесс в поддереву с некорректными результатами синтеза. В соответствии с реализацией обхода, сначала перебираются дети текущего поддерева и лишь затем перебор продолжается дальше. Данный метод выбран для исследования эффективности в силу своей простоты и наглядности. При реализации более продвинутых методов обхода дерева синтеза такой проблемы не должно возникать независимо от целевой функции.

### 4.3 Выводы

Новый метод способен конкурировать с существующей реализацией целевой функции в САПР NITTA несмотря на то, что при его реализации не использовалось никакой специализированной информации о предметной области. В проведённых испытаниях он показывал либо схожие, либо более предпочтительные в соответствии с обозначенными критериями результаты: время исполнения управляющей программы или глубина листа были меньше. Кроме того, в силу стохастического характера модели, она должна работать ещё лучше в комбинации с методами синтеза, которые способны брать в расчёт более одного успешного результата синтеза, что, предположительно, позволит смягчить влияние выбросов в ответах модели.

Стоит отметить, однако, что количества имеющихся в распоряжении примеров высокоуровневых алгоритмов и, соответственно, как разнообразия тренировочных данных, так и количества проведённых испытаний недостаточно для уверенной оценки. Для дальнейших исследований подхода существует необходимость реализовать больше различных примеров высокоуровневых алгоритмов для тренировки и испытания модели.

## ЗАКЛЮЧЕНИЕ

В настоящей работе произведён глубокий анализ проблемы высокой вычислительной сложности процесса синтеза, возникающей при разработке САПР специализированных вычислителей. Детально рассмотрены применяемые в предметной области абстракции, предложены эвристические методы оптимизации процесса синтеза, проанализирован существующий опыт решения аналогичных проблем в других областях. Поставленная задача была формализована для решения алгоритмами МО. Определён процесс обработки данных из САПР NITTA, предложена базовая модель. Спроектированное решение было реализовано. Эффективность полученной реализации исследована в соответствии с обозначенными при постановке задачи критериями.

Предложенный метод позволяет получить статистически обоснованную целевую функцию, с помощью которой реализуется эффективный алгоритм синтеза расписания управляющей программы. Таким образом, цель исследования считается достигнутой.

Однако, чтобы полностью раскрыть потенциал данного подхода, необходимы дальнейшие исследования. Рекомендуются реализовать больше примеров высокоуровневых алгоритмов для тренировки и исследования эффективности метода, а также оптимизировать выгрузку данных из САПР. Модели машинного обучения показывают результат тем достойнее, чем ближе распределение входных данных, на которых модели будут эксплуатироваться, к распределению входных данных тренировочного набора. Большое количество разнообразных тренировочных данных позволит получить модель, лучше обобщающуюся на реальные целевые алгоритмы. В свою очередь, большее количество экспериментов в рамках исследования эффективности уточнит оценку метода и увеличит степень уверенности в её корректности.

Кроме того, важным видится исследование применимости других моделей МО в данном формате: нейронных сетей с изменённой или принципиально другой архитектурой, моделей с байесовским подходом, случайного леса, других классических алгоритмов. Также, возможно, уместна дополнительная инженерия признаков или расширение входных данных дополнительной информацией.

Тем не менее, полученные результаты позволяют считать данную работу успешной проверкой концепции и мотивируют продолжать исследования в этой области по обозначенным направлениям.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Penskoi A.V. High-level synthesis system based on hybrid reconfigurable microarchitecture / Penskoi A.V., Platunov A.E., Kluchev A.O., Gorbachev Y.G., Yanalov R.I. // Scientific and Technical Journal of Information Technologies, Mechanics and Optics – 2019. – Т. 19 – № 2.
2. Nane R. A Survey and Evaluation of FPGA High-Level Synthesis Tools / Nane R., Sima V.M., Pilato C., Choi J., Fort B., Canis A., Chen Y.T., Hsiao H., Brown S., Ferrandi F., Anderson J., Bertels K. // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems – 2016. – Т. 35 – № 10.
3. Coulom R. Efficient selectivity and backup operators in Monte-Carlo tree search , 2007.
4. Chaslot G. Human-computer go revolution 2008 / Chaslot G., Hoock J.B., Rimmel A., Teytaud O., Lee C.S., Wang M.H., Tsai S.R., Hsu S.C. // ICGA Journal – 2008. – Т. 31 – № 3 – С.179–185.
5. Bouzy B. Associating shallow and selective global tree search with Monte Carlo for 9×9 Go , 2006.
6. Chaslot G.M.J.-B. Progressive Strategies for Monte-Carlo Tree Search / Chaslot G.M.J.-B., Winands M.H.M., Herik H.J. Van Den, Uiterwijk J.W.H.M., Bouzy B. // New Mathematics and Natural Computation – 2008. – Т. 04 – № 03.
7. Cotarelo A. Improving Monte Carlo Tree Search with Artificial Neural Networks without Heuristics / Cotarelo A., García-Díaz V., Núñez-Valdez E.R., González García C., Gómez A., Chun-Wei Lin J. // Applied Sciences – 2021. – Т. 11 – № 5.
8. Silver D. Mastering the game of Go with deep neural networks and tree search / Silver D., Huang A., Maddison C.J., Guez A., Sifre L., Driessche G. Van Den, Schrittwieser J., Antonoglou I., Panneershelvam V., Lanctot M., Dieleman S., Grewe D., Nham J., Kalchbrenner N., Sutskever I., Lillicrap T., Leach M., Kavukcuoglu K., Graepel T., Hassabis D. // Nature – 2016. – Т. 529 – № 7587.

9. Potdar K. A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers / Potdar K., S. T., D. C. // International Journal of Computer Applications – 2017. – Т. 175 – № 4.
10. Kingma D.P. Adam: A method for stochastic optimization , 2015.
11. Python Software Foundation Python 3.7.10 documentation// 15 February 2021 [Электронный ресурс]. URL: <https://docs.python.org/3.7/> (дата обращения: 16.05.2021).
12. Abadi M. TensorFlow: A system for large-scale machine learning , 2016.
13. McKinney W. Pandas - Powerful Python Data Analysis Toolkit// Pandas - Powerful Python Data Analysis Toolkit [Электронный ресурс]. URL: <https://pandas.pydata.org/docs/> (дата обращения: 16.05.2021).

## ПРИЛОЖЕНИЕ А

### РЕАЛИЗАЦИЯ СБОРЩИКА ДАННЫХ

```
import asyncio
import os
import pickle
import subprocess
import time
from dataclasses import dataclass, field
from pathlib import Path
from typing import List, Optional, Any

import pandas as pd
from aiohttp import ClientSession
from cached_property import cached_property
from cachetools import cached
from dataclasses_json import dataclass_json, LetterCase

NITTA_PORT = 53829
NITTA_BASEURL = f"http://localhost:{NITTA_PORT}"
NITTA_EXE_PATH = r'D:\Dev\nitta\build\nitta.exe'
NITTA_ROOT_DIR = r'D:\Dev\nitta'
METRICS_WEIGHTS = pd.Series(dict(duration=-1, depth=-0.1))
LAMBDA = 0.6

DATA_ROOT = Path("data")

DATA_ROOT.mkdir(exist_ok=True)

def cached_node_method(wrapped):
    return cached({}, key=lambda self, *args: hash(self.sid))(wrapped)

nitta_dataclass_params = dict(letter_case=LetterCase.CAMEL)

@dataclass_json(**nitta_dataclass_params)
@dataclass
class NittaNodeDecision:
    tag: str

@dataclass_json(**nitta_dataclass_params)
@dataclass
class NittaNode:
    score: Optional[int]
    is_terminal: bool
    is_finish: bool
    parameters: Any
    decision: NittaNodeDecision
    duration: Optional[int]
    sid: str
```

```

children: Optional[List['NittaNode']] = field(default=None, repr=False)
parent: Optional['NittaNode'] = field(default=None, repr=False)

@cached_property
def subtree_size(self):
    assert self.children is not None
    return sum(child.subtree_size for child in self.children) + 1

@cached_property
def depth(self) -> int:
    return self.sid.count('-') if self.sid != '-' else 0

@cached_property
def subtree_leafs_metrics(self) -> pd.DataFrame:
    if self.is_leaf:
        if not self.is_finish:
            return pd.DataFrame()
        return pd.DataFrame(dict(duration=[self.duration], depth=[self.depth]))
    else:
        return pd.concat([child.subtree_leafs_metrics for child in self.children])

@cached_node_method
def get_subtree_leafs_labels(self, metrics_distrib: pd.DataFrame) -> pd.Series:
    if self.is_leaf:
        return pd.Series([self.compute_label(metrics_distrib)])
    else:
        return pd.concat([child.get_subtree_leafs_labels(metrics_distrib) for child in
self.children])

@cached_node_method
def compute_label(self, metrics_distrib: pd.DataFrame) -> float:
    if self.is_leaf:
        if not self.is_finish:
            # unsuccessful synthesis, very low artificial label
            return -3

        metrics = self.subtree_leafs_metrics.iloc[0]
        normalized_metrics = (metrics - metrics_distrib.mean()) / metrics_distrib.std()
        return normalized_metrics.dot(METRICS_WEIGHTS)

    subtree_labels = self.get_subtree_leafs_labels(metrics_distrib)
    return LAMBDA * subtree_labels.max() + (1 - LAMBDA) * subtree_labels.mean()

@cached_property
def alternative_siblings(self) -> dict:
    bindings, refactorings, dataflows = 0, 0, 0

    if self.parent:
        for sibling in self.parent.children:
            if sibling.sid == self.sid:
                continue
            target = None
            if sibling.decision.tag == "BindDecisionView":
                bindings += 1
            elif sibling.decision.tag == "DataflowDecisionView":
                dataflows += 1

```



```

        else:
            refactorings += 1

    return dict(alt_bindings=bindings,
                alt_refactorings=refactorings,
                alt_dataflows=dataflows)

async def retrieve_subforest(self, session, levels_left=None):
    self.children = []
    if self.is_leaf or levels_left == -1:
        return

    async with session.get(NITTA_BASEURL + f"/node/{self.sid}/subForest") as resp:
        children_raw = await resp.json()

    for child_raw in children_raw:
        child = NittaNode.from_dict(child_raw)
        child.parent = self
        self.children.append(child)

    levels_left_for_child = None if levels_left is None else levels_left - 1
    await asyncio.gather(
        *[child.retrieve_subforest(session, levels_left_for_child) for child in
self.children]
    )

async def retrieve_whole_nitta_tree(max_depth=None) -> NittaNode:
    start_time = time.perf_counter()
    async with ClientSession() as session:
        async with session.get(NITTA_BASEURL + f"/node/-") as resp:
            root_raw = await resp.json()
            root = NittaNode.from_dict(root_raw)
            await root.retrieve_subforest(session, max_depth)

    print(f"Finished tree retrieval in {time.perf_counter() - start_time:.2f} s")
    return root

def _extract_params_dict(node: NittaNode) -> dict:
    if node.decision.tag in ["BindDecisionView", "DataflowDecisionView"]:
        result = node.parameters.copy()
        if node.decision.tag == "DataflowDecisionView":
            result["pNotTransferableInputs"] = sum(result["pNotTransferableInputs"])
        return result
    elif node.decision.tag == "RootView":
        return {}
    else:
        # refactorings
        return {"pRefactoringType": node.decision.tag}

def assemble_tree_dataframe(example: str, node: NittaNode, metrics_distrib=None,
include_label=True,
                            levels_left=None) -> pd.DataFrame:
    if include_label and metrics_distrib is None:
        metrics_distrib = node.subtree_leafs_metrics

```

```

self_df = pd.DataFrame(dict(
    example=example,
    sid=node.sid,
    tag=node.decision.tag,
    old_score=node.score,
    is_leaf=node.is_leaf,
    **node.alternative_siblings,
    **_extract_params_dict(node),
), index=[0])
if include_label:
    self_df["label"] = node.compute_label(metrics_distrib)

levels_left_for_child = None if levels_left is None else levels_left - 1
if node.is_leaf or levels_left == -1:
    return self_df
else:
    result = [assemble_tree_dataframe(example, child, metrics_distrib, include_label,
levels_left_for_child)
              for child in node.children]
    if node.sid != "-":
        result.insert(0, self_df)
    return pd.concat(result)

async def process_example(example: str) -> pd.DataFrame:
    example_name = os.path.basename(example)
    df = None

    print(f"Processing example {example!r}")
    with subprocess.Popen([NITTA_EXE_PATH, f'-p={NITTA_PORT}', example], cwd=NITTA_ROOT_DIR,
                          stdout=subprocess.PIPE, stderr=subprocess.STDOUT) as proc:
        try:
            for i in range(2):
                r = proc.stdout.readline()
                print(f"NITTA is running: ", r)
                time.sleep(2)
                print(f"Retrieving tree...")

            tree = await retrieve_whole_nitta_tree()
            with open(DATA_ROOT / f"{example_name}.pickle", "wb") as f:
                pickle.dump(tree, f)

            print(f"Nodes: {tree.subtree_size}. Building dataframe...")
            df = assemble_tree_dataframe(example_name, tree).reset_index(drop=True)

            print(f>Data's ready, {len(df)} rows")

            target_filepath = DATA_ROOT / f"{example_name}.csv"
            print(f>Saving to {target_filepath}")
            df.to_csv(target_filepath, index=False)
        finally:
            proc.kill()
            print(f"NITTA is dead")
    print("DONE")
    return df

```

## ПРИЛОЖЕНИЕ Б

### РЕАЛИЗАЦИЯ МОДЕЛИ МАШИННОГО ОБУЧЕНИЯ

```
from glob import glob
from pathlib import Path
from typing import Tuple

import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers

DATA_ROOT = Path("data")

MODELS_ROOT = Path("models")
MODELS_ROOT.mkdir(exist_ok=True)

data_csvs = glob(str(DATA_ROOT / "*.csv"))

TARGET_COLUMNS = ["label"]

def preprocess_df(df: pd.DataFrame) -> pd.DataFrame:
    def map_bool(c):
        return c.apply(lambda v: 1 if v is True else (0 if v is False else v))

    def map_categorical(df, c):
        return pd.concat([df.drop([c.name], axis=1), pd.get_dummies(c, prefix=c.name)],
axis=1)

    df = df.copy()
    df.is_leaf = map_bool(df.is_leaf)
    df.pCritical = map_bool(df.pCritical)
    df.pPossibleDeadlock = map_bool(df.pPossibleDeadlock)
    df.pRestrictedTime = map_bool(df.pRestrictedTime)
    df = map_categorical(df, df.tag)
    df = df.drop(["pWave", "example", "sid", "old_score", "is_leaf", "pRefactoringType"],
axis="columns")
    df = df.fillna(0)

    return df

def create_datasets(df) -> Tuple[tf.data.Dataset, tf.data.Dataset]:
    # create training and evaluation datasets
    train_df, test_df = train_test_split(df.sample(frac=1), test_size=0.2)

    N = len(df)
    print(f"N:\t{N}")
    print(f"Train:\t{len(train_df)}, {len(train_df) / N * 100:.0f}%")
    print(f"Test:\t{len(test_df)}, {len(test_df) / N * 100:.0f}%")

    def df_to_dataset(df, shuffle=True, batch_size=16, repeat=False, print_cols=False):
```

```

df = df.copy()

# split df into features and labels
targets = df[TARGET_COLUMNS].copy()
df.drop(TARGET_COLUMNS, axis=1, inplace=True)
features = df
if print_cols:
    print(f"Feature columns: {features.columns.values.tolist()}")

ds = tf.data.Dataset.from_tensor_slices((features.values, targets.values))
ds = ds.shuffle(buffer_size=10000) if shuffle else ds
ds = ds.batch(batch_size) if batch_size else ds
ds = ds.repeat() if repeat else ds
return ds

train_ds = df_to_dataset(train_df, batch_size=16, repeat=True, print_cols=True)
test_ds = df_to_dataset(test_df)

return train_ds, test_ds

def create_model() -> tf.keras.Model:
    model = tf.keras.Sequential([
        layers.Input(shape=(20,)),
        layers.Dense(128, activation="relu", kernel_regularizer="l2"),
        layers.Dense(128, activation="relu", kernel_regularizer="l2"),
        layers.Dense(64, activation="relu", kernel_regularizer="l2"),
        layers.Dense(64, activation="relu", kernel_regularizer="l2"),
        layers.Dense(32, activation="relu"),
        layers.Dense(1)
    ])

    model.compile(
        optimizer=tf.keras.optimizers.Adam(lr=3e-4),
        loss="mse",
        metrics=["mae"],
    )
    return model

df = pd.concat([pd.read_csv(d) for d in data_csvs]).reset_index(drop=True)
pdf = preprocess_df(df)
train_ds, test_ds = create_datasets(pdf)
model = create_model()
model.fit(
    train_ds,
    epochs=20,
    steps_per_epoch=2250,
    validation_data=test_ds
)
model.save(MODELS_ROOT / "model1")

```

## ПРИЛОЖЕНИЕ В

### РЕАЛИЗАЦИЯ ИССЛЕДОВАНИЯ ЭФФЕКТИВНОСТИ

```
from collections import Callable, defaultdict
from numbers import Number
from pathlib import Path
from typing import Optional

import pandas as pd
import tensorflow as tf
from aiohttp import ClientSession

from data_crawling import NittaNode, retrieve_whole_nitta_tree, assemble_tree_dataframe
from model_training import preprocess_df

Evaluator = Callable[[NittaNode], Number]

MODELS_ROOT = Path("models")

model = tf.keras.models.load_model(MODELS_ROOT / "model1")

final_columns = ['alt_bindings', 'alt_refactorings', 'alt_dataflows',
                 'pOutputNumber', 'pAlternative', 'pAllowDataFlow',
                 'pCritical', 'pPercentOfBindedInputs', 'pPossibleDeadlock',
                 'pNumberOfBindedFunctions', 'pRestless', 'pNotTransferableInputs',
                 'pRestrictedTime', 'pWaitTime', 'tag_BindDecisionView',
                 'tag_BreakLoopView', 'tag_ConstantFoldingView', 'tag_DataflowDecisionView',
                 'tag_OptimizeAccumView', 'tag_ResolveDeadlockView']
metrics_columns = [cn for cn in final_columns if cn.startswith("p")] + ["pRefactoringType",
"pWave"]

def new_evaluator(node: NittaNode) -> float:
    node_df = assemble_tree_dataframe("", node, include_label=False, levels_left=-1)
    filled_metrics_df = pd.concat([pd.DataFrame(columns=metrics_columns), node_df])
    preprocessed_df = preprocess_df(filled_metrics_df)
    right_final_columns_df = pd.concat([pd.DataFrame(columns=final_columns),
                                        preprocessed_df])[final_columns]
    ohe_flags_zero_filled_df = right_final_columns_df.fillna(0)
    final_df = ohe_flags_zero_filled_df

    return model.predict(final_df.values)[0][0]

def old_evaluator(node: NittaNode) -> int:
    return node.score

async def select_best_by_evaluator(session: ClientSession, evaluator: Evaluator, node:
NittaNode,
                                children_limit=None, call_counters=None) ->
Optional[NittaNode]:
    if call_counters is not None:
```

```

        call_counters[evaluator.__name__] += 1

    if node.is_leaf:
        return node if node.is_finish else None

    await node.retrieve_subforest(session, 0)
    children = [(evaluator(child), child) for child in node.children]
    children.sort(key=lambda v: v[0], reverse=True)
    if children_limit:
        children = children[:children_limit]

    while children:
        next_best_child = children.pop(0)[1]
        result = await select_best_by_evaluator(session, evaluator, next_best_child,
        children_limit, call_counters)
        if result is not None:
            return result

    return None

async def main():
    async with ClientSession() as session:
        root = await retrieve_whole_nitta_tree(-1)
        call_counters = defaultdict(lambda: 0)
        common_args = dict(
            session=session,
            node=root,
            children_limit=2,
            call_counters=call_counters
        )
        best_old = await select_best_by_evaluator(**common_args, evaluator=old_evaluator)
        best_new = await select_best_by_evaluator(**common_args, evaluator=new_evaluator)
    return best_old, best_new, call_counters

```