ИПМ им.М.В.Келдыша РАН

Абрау-2019 • Труды конференции

С.А. Гречаник

Перспективы применения нейронных сетей в суперкомпиляции и насыщении равенствами

Рекомендуемая форма библиографической ссылки

Гречаник С.А. Перспективы применения нейронных сетей в суперкомпиляции и насыщении равенствами // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 278-289. — URL: http://keldysh.ru/abrau/2019/theses/35.pdf doi:10.20948/abrau-2019-35

Размещена также презентация к докладу

Перспективы применения нейронных сетей в суперкомпиляции и насыщении равенствами

С.А. Гречаник

ИПМ им.М.В. Келдыша РАН

Аннотация. Многие методы преобразования программ (включая суперкомпиляцию и насыщение равенствами) можно сформулировать в виде набора правил переписывания графов или термов, применяемых в некотором порядке, заданном эвристиками. Эти эвристики обычно создаются вручную, поэтому интересно автоматизировать их создание, например, при помощи машинного обучения. В данной работе мы дадим обзор некоторых подходов к решению подобных задач при помощи нейронных сетей.

Ключевые слова: машинное обучение, нейронные сети, анализ программ, преобразование программ

Prospects of applying neural networks to supercompilation and equality saturation

S.A.Grechanik

Keldysh Institute of Applied Mathematics

Abstract. Many methods of program transformation (including supercompilation and equality saturation) may be formulated as a set of term or graph rewriting rules that are applied in the order defined by heuristics. These heuristics are usually created by hand, so it is very interesting to automate their creation, e.g. with machine learning. In this paper we survey some approaches to tackle these problems with neural networks.

Keywords: machine learning, neural networks, program analysis, program transformation

1. Введение

Во многих методах преобразования программ возникает проблема управления порядком применения преобразований. Например, при переписывании термов результат может сильно зависеть от порядка применения правил переписывания. Наилучший подход – преобразовать систему правил переписывания таким образом, чтобы результат не зависел от

порядка применения правил, однако это не всегда возможно, и тогда приходится применять эвристики.

В суперкомпиляции [26] также возникает эта проблема – на каждом шаге построения графа конфигураций можно применить различные шаги: прогонку, обобщение, зацикливание. То, какой шаг будет применён, управляется эвристиками, в частности, свистком, который указывает, в каком месте вместо прогонки надо применить обобщение.

Существуют попытки избежать эвристик. Например, насыщение равенствами [25] состоит в том, чтобы применять правила во всех возможных порядках, преобразуя не один терм или программу, а целое множество термов или программ. Это приводит к комбинаторному взрыву, с которым частично справляется совмещение общих частей различных программ или термов. В многорезультатной суперкомпиляци также разрешается одновременное исследование различных путей преобразования программы (однако эвристики всё же используются).

В работе [11] производится совмещение методов суперкомпиляции и насыщения равенствами. Несмотря на то, что насыщение равенствами способно бороться с комбинаторным взрывом, для правил переписывания, взятых из суперкомпиляции, этого оказывается недостаточно, и приходится применять эвристики, а именно среди правил были выделены упрощающие правила, которые нужно применять на каждом шаге до достижения нормальной формы. Кроме того, некоторые правила применяются деструктивно, в отличие от оригинального насыщение равенствами. То, какие правила считать упрощающими и какие применять деструктивно, определялось вручную.

Таким образом, эвристики являются необходимыми, и возникает вопрос, можно ли автоматизировать их создание. Последние годы набрали популярность нейронные сети, главным образом благодаря успехам в области анализа изображений и обработки текста на естественном языке. Для анализа программ нейронные сети также применяются, поэтому интересно исследовать вопрос построения эвристик для насыщения равенствами и суперкомпиляции на основе нейронных сетей.

В настоящее время существует очень большое количество архитектур нейронных сетей, поэтому здесь мы будем подразумевать под нейронной сетью просто функцию, параметризованную набором действительных чисел $f(x; \theta)$, $\theta \in \mathbb{R}^n$. Параметр θ для удобства часто опускают. Входы и выходы нейронных сетей могут быть какими угодно, например, в случае анализа программ входом зачастую является программа, представленная в виде дерева или графа. Выходом обычно является действительное число или вектор действительных чисел, однако это необязательно.

Обучение нейронной сети заключается в поиске наилучшего набора параметров θ и обычно производится методом градиентного спуска. Наиболее распространённый подход – обучение с учителем, когда для некоторой выборки входов $\{x_i\}$ известны соответствующие выходы $\{y_i\}$. Тогда необходимо задать

функцию потерь loss(x, y), оценивающую, насколько близки выходы нейронной сети к эталонным, такую, что композиция loss(f(x; θ), y) дифференцируема по θ. В случае обучения эвристик, однако, таких эталонных данных для обучения может не быть, из-за чего приходится использовать более сложные методы, такие как обучение с подкреплением.

В данной статье мы рассмотрим некоторые работы последних лет, в которых нейронные сети применяются для анализа и преобразования программ, а также для некоторых задач, родственных суперкомпиляции и насыщению равенствами (доказательство теорем, комбинаторные оптимизации и т.п.).

Статья организована следующим образом. В разделе 2 даётся обзор работ, связанных с применением нейронных сетей для анализа программ, причём работы классифицированы в соответствии со способом представления программ (в разделе 2.1 рассматриваются работы, представляющие программы как последовательности токенов, в разделе 2.2 — как деревья, в разделе 2.3 — как графы). Далее в разделе 3 даётся обзор работ, в которых нейронных сети используются для решения различных комбинаторных задач (связанных не только с преобразованием программ). В разделе 4 описываются некоторые возможные подходы к применению нейронных сетей для суперкомпиляции и насыщения равенствами. Раздел 5 является заключением.

2. Анализ программ и выражений при помощи нейронных сетей

Сначала рассмотрим работы, в которых нейронные сети используются для анализа программ. Рассматриваемые работы можно классифицировать по тому, в каком виде программа подаётся на вход нейронной сети (или наоборот, извлекается на выходе). Основные походы следующие: рассматривать программу как последовательность (токенов или символов), рассматривать программу как дерево (AST) и рассматривать программу как граф (например, граф потока управления).

2.1 Анализ программ-последовательностей

Анализ программ, представленных в виде последовательности символов или токенов возможен, однако при этом нейронной сети приходится учиться выявлять синтаксическую структуру самостоятельно, что не эффективно, поэтому в настоящее время данный подход выглядит не очень перспективным. Тем не менее, рассмотрим несколько работ, использующих этот подход.

Часто (хоть и не всегда) для анализа последовательностей применяются рекуррентные нейронные сети (RNN, Recurrent Neural Networks). В основе рекуррентной нейронной сети лежит композиция функций вида $f([x_1 , ... , x_n]) = g(x_n , g(x_{n-1} , ... , g(x_1, s_0) ...))$, где $[x_1 , ... , x_n]$ — входная последовательность, s_0 — начальное состояние (обычно состояние — вектор действительных чисел), а g — некоторая функция, которое берёт на вход очередной элемент и старое состояние и возвращает новое состояние (и часто также и некоторый выходной

вектор). В простейшем случае g может иметь вид $g(x, s) = \sigma(Wx + Us + b)$, где W, U и b — тренируемые параметры, однако современные рекуррентные нейронные сети, почти всегда строятся на основе LSTM (long short-term memory [12]) или GRU (gated recurrent unit [5]), которые лучше учатся. Отметим также, что во многих работах в дополнении к рекуррентным сетям применяется механизм внимания (attention) [2].

В работе Cummins et al., “End-to-End Deep Learning of Optimization Heuristics”, 2017 [6] нейронные сети применяются для оптимизации кода на OpenCL, а именно для двух задач: выбора наилучшего устройства для исполнения ядра и выбор степени слияния потоков (coarsening factor). В качестве архитектуры применяются два слоя LSTM, с двумя полносвязными блоками сверху. На вход подаётся исходный код, слегка нормализованный (стандартизованы имена переменных и т.п.), представленный как последовательность токенов. На выходе получается вектор, каждый элемент которого соответствует параметру оптимизации (устройству или значению степени слияния) и выражает степень уверенности нейронной сети, что именно этот параметр оптимизации подходит наилучшим образом к данной программе (соответственно далее при оптимизации кода нужно использовать параметр с наибольшей степенью уверенности). Для обучения данной модели используются программы, для которых известны оптимальные параметры оптимизации.

В работе Shin et al., “Towards Specification-Directed Program Repair”, 2018 [22] также используются рекуррентные нейронные сети на программах, представленных последовательностями токенов, но для другой задачи — исправления ошибок в программах. Для этого на вход нейронной сети вместе с программой подаётся набор пар вход-выход, а на выходе генерируется исправленная программа, которая должна удовлетворять этим парам. Входная программа анализируется при помощи LSTM, но в результате превращается не в один вектор, а в последовательность векторов, длина этой последовательности равна длине программы. Выходная программа генерируется также при помощи LSTM, причём в работе описано два варианта архитектуры: первая генерирует последовательно новую программу, при этом информация об исходной программе подаётся при помощи механизма внимания; вторая генерирует не программу, а изменения относительно исходной программы (удаление, вставка и замена операций). Вторая архитектура оказалась несколько хуже, но она быстрее для тренировки благодаря тому, что не используется механизм внимания. Для обучения модели в этой работе использовался набор корректных программ, которые потом портились небольшими случайными изменениями.

2.2 Анализ программ-деревьев

Для программ и выражений естественным является представление в виде абстрактных синтаксических деревьев. Деревья можно анализировать при

помощи рекурсивных нейронный сетей [10]. Они похожи на рекуррентные сети, но сворачивают не последовательности, а деревья, т.е. к каждому узлу применяется функция $g(x, s_1, ..., s_n)$, где x — метка узла, а s_i — представления, полученные применением этой же нейронной сети рекурсивно к дочерним узлам. Опять же, данная функция часто строится на основе LSTM (такая архитектура называется TreeLSTM [24]).

В работе Chen et al., “Tree-to-tree Neural Networks for Program Translation”, 2018 [4] рассматривается задача трансляции с одного языка программирования на другой (например, с Java на C#). Исходная программа превращается в синтаксическое дерево, которое бинаризуется и подаётся на вход кодировщику — рекурсивной нейронной сети на основе LSTM, в результате работы которой каждому узлу дерева ставится в соответствие состояние — вектор фиксированной длины. Затем декодировщик — вторая нейронная сеть на основе LSTM — порождает результирующую программу в виде дерева. В процессе генерации информация об исходной программе поступает при помощи механизма внимания, а кроме того корневое состояние кодировщика становится исходным состоянием декодировщика. В работе также рассматриваются и некоторые другие модели, в которых кодировщик или декодировщик заменяются на рекуррентные нейронные сети, рассматривающие программы как последовательности. Оказывается, что такие архитектуры работают хуже. Кроме того, экспериментально показывается, что механизм внимания критически важен для данной задачи.

В работе Zaremba et al., “Learning to Discover Efficient Mathematical Identities”, 2014 [28] рассматривается задача оптимизации простых математических выражений над матрицами (критерий оптимизации — вычислительная сложность, которая считается по выражению напрямую без тестирования). При этом также используется рекурсивная нейронная сеть на деревьях разбора выражений. Эта сеть сначала предобучается на задаче классификации выражений: генерируются все выражения до некоторой глубины, потом эти выражения разбиваются на классы эквивалентности при помощи тестирования, затем сеть учится предсказывать класс эквивалентности выражения. Потом эта сеть дообучается для решения задачи предсказания следующего правила в процессе построения выражения. Отметим, что в данной работе рекурсивная нейронная сеть зачастую проигрывает более простым методам.

2.3 Анализ программ-графов

Следующий шаг — анализ программ и выражений, представленных в виде графа. Представление в виде графа позволяет закодировать больше информации о программе, например можно превратить программу или выражение в граф взяв за основу AST и добавив дуги, соответствующие различным синтаксическим и семантическим связям (например, связать дугами все идентификаторы обозначающие одну и ту же сущность).

Для анализа графов предназначены графовые нейронные сети, видов которых существует очень много [27]. Для анализа программ часто применяется подход, основанный на обмене сообщениями между узлами графа, связанными рёбрами [9]. Этот подход заключается в том, что каждому узлу i сопоставляется последовательность состояний $h_{i,t} \in \mathbb{R}^n$, t = 0, ..., m, обновление состояний происходит по формуле $h_{i,t+1} = U(h_{i,t}, \sum_{(i,j)\in G} M(h_{i,t}, h_{j,t}, e_{i,j}))$, т.е. сначала для каждого ребра вычисляется сообщение при помощи функции M (сообщение зависит от состояний узлов и метки ребра), а затем все сообщения суммируются и происходит обновление состояния узла при помощи функции U (обе функции как обычно параметризованы некоторыми обучаемыми параметрами). Часто U реализована с помощью LSTM или GRU [3]. Весь процесс обычно повторяется некоторое фиксированное число шагов, хотя существуют работы, в которых его повторяют до сходимости [20].

В работе Allamanis et al., “Learning to Represent Programs with Graphs”, 2017 [1] рассматриваются задачи предсказания имён переменных в программах и предсказания неправильно использованных переменных. При этом используются нейронные сети на графах. Программы представляются в виде графов, которые в целом основаны на AST, но содержат дополнительные рёбра, которые соединяют терминальные вершины в последовательность, а также рёбра, описывающие информацию о потоке управления. Нейронная сеть устроена именно так как описано выше: каждой вершине ставится в соответствие вектор-состояние, затем для каждого ребра из вершины v в вершину u считается сообщение — вектор, полученный умножением состояния вершины v на матрицу весов. Далее все сообщения отправляются вдоль рёбер, и для каждой вершины u все входящие сообщения суммируются и используются вместе со старым состоянием вершины u для вычисления нового состояния при помощи блока GRU. Этот обмен сообщениями повторяется 8 раз, и конечные состояния вершин используются для вычисления предсказаний.

Отметим также работу Selsam et al., “Learning a SAT Solver from Single-Bit Supervision”, 2018 [21], в которой используются нейронные сети на графах для предсказания выполнимости булевых формул (в виде КНФ). Для этого по формуле строится граф: каждой дизъюнкции литералов ставится в соответствие вершина, и каждому литералу (переменная или отрицания) также ставится в соответствие вершина. Рёбра соединяют вершину-литерал с вершиной-дизъюнкцией, если этот литерал входит в дизъюнкцию. Переменные и их отрицания также соединены рёбрами, но другого вида (т.е. с другими весами). Нейронная сеть, работающая на этом графе, устроена примерно так же, как и в предыдущей работе: каждой вершине сопоставляется состояние-вектор, потом вершины обмениваются сообщениями-векторами и обновляют свои состояния. После нескольких итераций информация из состояний всех вершин-литералов суммируется, и на выходе получается одно число, означающее степень уверенности сети, что формула выполнима. Тренировка этой модели осуществляется на случайных формулах, причём формулы подбираются такими

парами, что одна формула выполнима, а другая нет, но отличаются они только одним литералом. Это сделано для того, чтобы модель не зацепилась за какие-либо статистические аномалии в структуре формул, а вместо этого изобрела некий алгоритм решения этой задачи, основанный на обмене сообщениями. Анализ, приведённый в работе позволяет предположить, что так оно и произошло.

3. Обучение эвристик

В этом разделе мы рассмотрим работы, заключающиеся в обучении эвристик для разных комбинаторных задач, причём сосредоточимся именно на методах обучения и том, как эвристики встроены в процесс поиска, а не на архитектурах нейронных сетей, лежащих в основе этих эвристик.

Для обучения нейронной сети при помощи обучения с учителем требуется набор пар вход-выход. В случае эвристик такого эталонного набора скорее всего нет. Один из вариантов решения этой проблемы — сгенерировать этот набор (что может быть вычислительно очень затратно). Другой вариант — использовать обучение с подкреплением, в котором генерация данных и обучение эвристики будут идти одновременно: улучшение эвристики приводит к сбору лучших (более точных) данных, которые в свою очередь позволяют лучше обучить эвристику.

В качестве примера первого подхода можно привести работу Edelmann et al., “Neural-Network Guided Expression Transformation”, 2019 [8], в которой решается задача поиска последовательности применений правил переписывания к арифметическому выражению для получения другого выражения (оба выражения подаются на вход). Для этого используется алгоритм A*, а в качестве эвристики используется нейронная сеть, основанная на TreeLSTM, берущая на вход два выражения и возвращающая оценку расстояния между ними (под расстоянием между выражениями понимается минимальное количество шагов переписывания, необходимых чтобы превратить одно в другое). Эвристика тренируется при помощи обучения с учителем, данные для обучения генерировались в течение нескольких недель.

В области обучения с подкреплением один из самых известных результатов за последние годы — достижение превосходства над человеком в игре Го программой AlphaGo. В работе Silver et al., “Mastering the game of go without human knowledge”, 2017 [23] описывается улучшенная версия, AlphaGo Zero, обучаемая исключительно за счёт игры с самой собой. В этой работе используется MCTS (Monte-Carlo Tree Search) совместно с нейронной сетью, берущей на вход текущее состояние и возвращающей оценку выигрышности текущей позиции и стратегию (policy), представленную вектором, описывающим вероятности всех ходов из данного состояния. В целом обучение ведётся при помощи алгоритма, близкого к policy iteration, заключающегося в том, что при помощи MCTS сначала улучшается стратегия на основе оценок выигрышности позиций, возвращаемых нейронной сетью, а затем улучшается

оценка выигрышности путём проведения игры до конца с использованием улучшенной стратегии. Хотя подход AlphaGo Zero предназначен в первую очередь для игр, существуют работы, применяющие его для решения комбинаторных задач, например, для раскраски графов [13].

В работе Dai et al., “Learning Combinatorial Optimization Algorithms over Graphs”, 2017 [7] нейронные сети используются для приближённого решения NP-трудных задач на графах (таких, как задача коммивояжёра). Подход основан на жадном алгоритме: на каждом шаге графовая нейронная сеть оценивает все вершины графа, и к решению добавляется вершина с наилучшей оценкой. Для тренировки нейронной сети используется Q-обучение (Q-learning, через $Q^{\pi}(s, a)$ обычно обозначают функцию, возвращающую суммарное подкрепление если в состоянии s произвести действие a, а затем следовать стратегии π). Нейронная сеть при этом фактически является текущей оценкой функции Q. Упрощая, Q-обучение также состоит в том, чтобы производить действие с максимальным значением Q (точнее, с некоторой вероятностью вместо этого может производиться случайное действие для разведки), попутно собирая данные о подкреплении для корректировки нейронной сети.

В работе Li et al., “Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search”, 2018 [17] также решаются NP-трудные задачи на графах (главным образом, задача о максимальном независимом множестве вершин, остальные сводятся к ней). В отличие от предыдущей работы, в этой работе вместо обучения с подкреплением используется обучение с учителем, а данные генерируются с помощью классических решателей. В работе представлены два подхода. В первом также используется жадный алгоритм, управляемый графовой нейронной сетью. Нейронная сеть на выходе выдаёт для каждой вершины вероятность того, что она принадлежит решению, и для её обучения используется стандартная функция потерь на основе перекрёстной энтропии. Второй подход отличается тем, что нейронная сеть выдаёт набор из нескольких возможных вероятностей для каждой вершины, и в качестве функции потерь берётся минимум перекрёстных энтропий. Для генерации решения используется более сложный рандомизированный алгоритм поиска, строящий несколько решений. Этот подход требуется для увеличения разнообразия решений. Также в этой работе используются классические алгоритмы улучшения уже построенного решения путём локальных преобразований.

Существует также много работ, посвящённых применению нейронных сетей в доказательстве теорем. Например, в работе Kusumoto et al., “Automated Theorem Proving in Intuitionistic Propositional Logic by Deep Reinforcement Learning”, 2018 [16] описывается подход к доказательству теорем в интуиционистской логике. В ней используется обучение с подкреплением (а именно алгоритм approximate policy iteration). В качестве обучаемой эвристики используется графовая нейронная сеть, которая возвращает оценку ожидаемого

выигрыша для данного ей на вход состояния (представляющего из себя множество секвенций).

В более ранней работе Loos et al., “Deep Network Guided Proof Search”, 2017 [18] нейронная сеть используется для выбора следующего дизъюнкта для обработки в системе доказательства теорем E. При этом, однако, используется обучение с учителем, а именно нейронная сеть тренируется предсказывать, используется ли данный ей на вход дизъюнкт в доказательстве данного ей на вход утверждения.

В работе Kaliszyk et al., “Reinforcement Learning of Theorem Proving”, 2018 [14] описывается обучение эвристик при помощи MCTS для табличного метода доказательства теорем. Эвристики, впрочем, основаны не на нейронных сетях, а на бустинге и признаках, сконструированных вручную.

4. Возможные применения в суперкомпиляции и насыщении равенствами

Данный раздел носит спекулятивный характер, поскольку ни одной работы, применяющей нейронные сети в рамках суперкомпиляции или насыщения равенствами автору не известно.

В случае классической суперкомпиляции самый очевидный компонент, который можно было бы реализовать при помощи нейронной сети — свисток. Свисток может выдавать 0 или 1, когда свисток выдаёт 1 (свистит), вместо правил прогонки применяются правила обобщения. Есть разные виды свистков, простейший — унарный, получающий на вход конфигурацию (выражение), находящуюся в текущей вершине. Такой свисток можно реализовать при помощи рекурсивных или графовых нейронных сетей. Для обучения скорее всего необходимо использовать обучение с подкреплением. Отметим, что классическая суперкомпиляция похожа на системы доказательства теорем, поэтому ожидается, что подойдут методы, применяемые в этой области.

Более интересным случаем является насыщение равенствами. Дело в том, что насыщение равенствами похоже на переписывание термов, однако благодаря недеструктивности принятие ошибочного решения в выборе следующего правила переписывания не является катастрофическим, поскольку не запрещает применять “конкурирующие” правила. Это означает, в частности, что насыщение равенствами должно хорошо работать с рандомизированными стратегиями. Кроме того, существуют методы, позволяющие из конечного состояния получить цепочку применений правил, которая привела к некоторому терму (без лишних правил) [19]. Поэтому мы предлагаем такой подход к обучению: на очередном шаге обучения применить правила в соответствии с некоторой стратегией (скорее всего полученной на предыдущем шаге, но с некоторым шумом), затем выделить из получившегося графа наилучший терм (программу) вместе с приведшей к нему цепочкой применений правил и поправить параметры так, чтобы увеличить вероятности правил из цепочки. Так как насыщение равенствами выполняет преобразования над

графом (точнее, E-графом, который можно представить в виде двудольного графа), то логично применить графовые нейронные сети. Работа в этом направлении ведётся автором, но пока далека от завершения.

5. Заключение

Мы рассмотрели некоторые работы, посвящённые анализу программ при помощи нейронных сетей. Из всех подходов наиболее перспективным в настоящий момент кажется применение графовых нейронных сетей. Мы также рассмотрели работы, посвящённые решению различных задач при помощи обучаемых эвристик. Многие работы используют обучение с подкреплением, но многие — обучение с учителем, и пока сложно сделать вывод, какой из них предпочтительнее.

Суперкомпиляция довольно близка к задаче доказательства теорем, поэтому видится перспективным применение подходов из этих работ для суперкомпиляции. В насыщение равенствами также можно ожидать успешного применения эвристик, основанных на графовых нейронных сетях.

Работа выполнена при поддержке гранта РФФИ № 18-31-00412.

References

1. Allamanis M., Brockschmidt M., Khademi M. Learning to Represent Programs with Graphs // CoRR. — 2017. — Vol. abs/1711.00740. — URL: http://arxiv.org/abs/1711.00740.
2. Bahdanau D., Cho K., Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate // ICLR / ed. by Y. Bengio, Y. LeCun. — 2015. — URL: http://arxiv.org/abs/1409.0473.
3. Beck D., Haffari G., Cohn T. Graph-to-Sequence Learning using Gated Graph Neural Networks // CoRR. — 2018. — Vol. Abs/1806.09835. — URL: http://arxiv.org/abs/1806.09835.
4. Chen X., Liu C., Song D. Tree-to-tree Neural Networks for Program Translation // CoRR. — 2018. — Vol. abs/1802.03691. — URL: http://arxiv.org/abs/1802.03691.
5. Cho K., Merrienboer B. van, Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y. Learning Phrase Representations using RNN Encoder-Decode for Statistical Machine Translation. — 9 02/2014. — URL: http://arxiv.org/abs/1406.1078.
6. Cummins C., Petoumenos P., Wang Z., Leather H. End-to-End Deep Learning of Optimization Heuristics // PACT. — IEEE Computer Society, 2017. — P. 219–232. — ISBN 978-1-5090-6764-0.
7. Dai H., Khalil E. B., Zhang Y., Dilkina B., Song L. Learning Combinatorial Optimization Algorithms over Graphs // CoRR. — 2017. — Vol. Abs/1704.01665. — URL: http://arxiv.org/abs/1704.01665.

8. Edelmann R., Kuncak V. Neural-Network Guided Expression Transformation // CoRR. — 2019. — Vol. abs/1902.02194. — URL: http://arxiv.org/abs/1902.02194.
9. Gilmer J., Schoenholz S. S., Riley P. F., Vinyals O., Dahl G. E. Neural Message Passing for Quantum Chemistry // CoRR. — 2017. — Vol. Abs/1704.01212. URL: http://arxiv.org/abs/1704.01212.
10. Goller C., Kuchler A. Learning task-dependent distributed structure-representations by backpropagation through structure // IEEE International Conference on Neural Networks. — 1996. — P. 347–352.
11. Grechanik S. Inductive Prover Based on Equality Saturation (Extended Version) // Proceedings of the Fourth International Valentin Turchin Workshop on Metacomputation / ed. by A. Klimov, S. Romanenko. — Pereslavl-Zalessky, Russia : Pereslavl Zalessky: Publishing House ”University of Pereslavl”, 07/2014. — P. 26–53. — ISBN 978-5-901795-31-6.
12. Hochreiter S., Schmidhuber J. Long Short-Term Memory // Neural Computation. — 1997. — Vol. 9, no. 8. — P. 1735–1780.
13. Huang J., Patwary M. M. A., Diamos G. F. Coloring Big Graphs with AlphaGoZero // CoRR. — 2019. — Vol. abs/1902.10162. — URL: http://arxiv.org/abs/1902.10162.
14. Kaliszyk C., Urban J., Michalewski H., Olsák M. Reinforcement Learning of Theorem Proving // CoRR. — 2018. — Vol. abs/1805.07563. — URL: http://arxiv.org/abs/1805.07563.
15. Klyuchnikov I. G., Romanenko S. A. Multi-result Supercompilation as Branching Growth of the Penultimate Level in Metasystem Transitions // Ershov Memorial Conference. Vol. 7162 / ed. by E. M. Clarke, I. Virbitskaite, A. Voronkov. — Springer, 2011. — P. 210–226. — (Lecture Notes in Computer Science). — ISBN 978-3-642-29708-3.
16. Kusumoto M., Yahata K., Sakai M. Automated Theorem Proving in Intuitionistic Propositional Logic by Deep Reinforcement Learning // CoRR. — 2018. — Vol. abs/1811.00796. — URL: http://arxiv.org/abs/1811.00796.
17. Li Z., Chen Q., Koltun V. Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search // CoRR. — 2018. — Vol. Abs/1810.10659. — URL: http://arxiv.org/abs/1810.10659.
18. Loos S. M., Irving G., Szegedy C., Kaliszyk C. Deep Network Guided Proof Search // CoRR. — 2017. — Vol. abs/1701.06972. — URL: http://arxiv.org/abs/1701.06972.
19. Nieuwenhuis R., Oliveras A. Proof-producing congruence closure // International Conference on Rewriting Techniques and Applications. — Springer. 2005. — P. 453–468.
20. Scarselli F., Gori M., Tsoi A. C., Hagenbuchner M., Monfardini G. The Graph Neural Network Model // IEEE Trans. Neural Networks. — 2009. — Vol. 20, no. 1. — P. 61–80.

21. Selsam D., Lamm M., Bünz B., Liang P., Moura L. de, Dill D. L. Learning a SAT Solver from Single-Bit Supervision // CoRR. — 2018. — Vol. Abs/1802.03685. — URL: http://arxiv.org/abs/1802.03685.
22. Shin R., Polosukhin I., Song D. Towards Specification-Directed Program Repair. — 2018
23. Silver D., Schrittwieser J., Simonyan K., Antonoglou I., Huang A., Guez A., Hubert T., Baker L., Lai M., Bolton A., et al. Mastering the game of go without human knowledge // Nature. — 2017. — Vol. 550, no. 7676. — P. 354.
24. Tai K. S., Socher R., Manning C. D. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks // ACL (1). — The Association for Computer Linguistics, 2015. — P. 1556–1566. — ISBN 978-1-941643-72-3.
25. Tate R., Stepp M., Tatlock Z., Lerner S. Equality saturation: a new approach to optimization // SIGPLAN Not. — New York, NY, USA, 2009. — Jan. — Vol. 44, issue 1. — P. 264–276. — ISSN 0362-1340. — URL: http://doi.acm.org/10.1145/1594834.1480915.
26. Turchin V. The concept of a supercompiler // ACM Transactions on Programming Languages and Systems (TOPLAS). — 1986. — Vol. 8, no. 3. — P. 292–325.
27. Wu Z., Pan S., Chen F., Long G., Zhang C., Yu P. S. A Comprehensive Survey on Graph Neural Networks // CoRR. — 2019. — Vol. Abs/1901.00596. — URL: http://arxiv.org/abs/1901.00596.
28. Zaremba W., Kurach K., Fergus R. Learning to Discover Efficient Mathematical Identities // CoRR. — 2014. — Vol. abs/1406.1584. — URL: http://arxiv.org/abs/1406.1584.