

# Обзор GOLEM

Владимир Латыпов

09-02-2024

Дисклеймер: презентация больше адаптирована для рассказа с периодическим переходом на демонстрацию экрана со статьями, issues или кодом, но на свой страх и риск можно и попробовать её читать без докладчика.

# История появления

Изначально была библиотека **FEDOT** для AutoML, основана на pipeline-ах ~произвольной структуры (dag), поиск происходит посредством эволюции. Но алгоритм графовой оптимизации оказался полезен и для кучи других задач, в т.ч. проектов лаборатории:

- BAMT (Bayesian AutoML Tool)
- NAS (Neural Architecture Search)
- GEFEST (Generative Evolution For Encoded STructures)
- пользовательские применения (коллаборация с химической лабораторией, например — btw полезный подход)

Поэтому было решено выделить эту часть в отдельную библиотеку — GOLEM.

# Возможности FEDOT

		STREAMLINE	Auto-Keras	H2O-3 AutoML	MLme	LAMA	FEDOT	FLAML	ALIRO	PYCARET	Auto-Gluon	MLJAR-supervised	Ludwig	TPOT	Auto-Sklearn	Auto-PyTorch	GAMA	Hyperopt-sklearn	Auto-WEKA	RECIPE	ML-Plan	TransmogriAI	MLBox	Xcessiv	Auto_ML
		YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Target	Binary Classification	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	MultiClass Classification	NO	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES		YES	YES	YES	YES	YES
	Regression	NO	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	NO	YES	YES	YES	YES	YES
	Multi-Task	NO	YES	NO	NO	NO	NO	NO	NO	NO	NO	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
	Multi-Label	NO	NO	NO	NO	NO	NO		NO	NO	NO	NO	NO	NO	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
	Clustering	NO	NO	NO	NO	NO	YES	NO	NO	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
	Anomaly Detection	NO	NO	NO	NO	NO	NO	NO	NO	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO

# Глобально про алгоритм

Структура — сложный гиперпараметр, так у самого задания структуры нет фиксированной структуры...

Использует эволюцию для оптимизации над графами, в том числе — многокритериальной. Предоставляет многие стандартные операторы, есть интерфейс для добавления пользовательских.

# Операторы

Глобально — то, что приоткрывает нам чёрнощиковость представления генома и преобразует одни геномы в другие.

Эволюция — последовательное применение этих операторов.

Ожидается их «семантичность»: например, то, что мы можем получать небольшие изменения структуры с точки зрения задачи → это отражается и на значениях фитнес функции.

Пример несемантичности: если кодировать вещественнозначные решения битами, получится плохо.

# Мутации

Мутации и кроссовер учитывают `GraphGenerationRequirements`, а также `GraphVerifier`.

Мутации, семантические для графов:

- Заменить атрибут каждой ноды с заданной вероятностью на то, что сгенерирует `NodeFactory` и так, как посоветует `ChangeAdvisor` (например, в химии некоторые связи невозможны)
- Добавление ребра между случайными нодами
- Вставка случайной ноды в разрыв ребра
- Заменить/выбросить ноду
- Заменить поддереву ноды на случайное
- ...пользовательские, особенно семантические

# Кроссовер

- Замена случайных поддеревьев
- Нахождение структурно эквивалентных подграфов и замена случайных нод или поддеревьев в них
- Смена родителями с аналогичной нодой в другом графе
- ... пользовательские



# Селекция

1. Tournament(fraction): каждый раз из группы размера  $\approx \text{population\_size} * \text{fraction}$  выбирается лучший в итоговую популяцию и убирается из кандидатов.
2. SPEA-2

Strength — количество особей, которые доминирует заданная:

$$S(i) = |\{j \mid j \in P_t + \overline{P}_t \wedge i \succ j\}|$$

Тогда назначаем raw fitness-ом сумму strengths всех, кого особь доминирует (то есть особо выгодно доминировать крутых):

$$R(i) = \sum_{j \in P_t + \overline{P}_t, i \succ j} S(j)$$

Но ещё хотим учитывать разнообразие. Про пространство оптимизации в общем случае ничего не знаем, поэтому считаем разнообразие в пространстве objective functions:

$$R(i) = \frac{1}{\sigma_i^k + 2}$$

, где  $\sigma_i^k$  — расстояние  $k$ -го ближайшего среди популяции + архива, а  $k$  выбирают  $\sqrt{N + \bar{N}}$

- Селекция новой популяции: производим селекцию турниром по  $F$  с заменой (участников турнира убираем из кандидатов)
- Селекция архива: выбираем недоминированных, но если не соответствует целевому размеру:

↳ Слишком большой: truncat-им по фитнесу (раз недоминированные, эквивалентно, по плотности): на каждом шагу убираем лексикографически меньшего по вектору расстояний до  $k$ -го ближайшего:

$$i \leq_d j \quad :\Leftrightarrow \quad \forall 0 < k < |\overline{\mathbf{P}}_{t+1}| : \sigma_i^k = \sigma_j^k \quad \vee \\ \exists 0 < k < |\overline{\mathbf{P}}_{t+1}| : \left[ \left( \forall 0 < l < k : \sigma_i^l = \sigma_j^l \right) \wedge \sigma_i^k < \sigma_j^k \right]$$

↳ Слишком маленький: добавляем оставшихся сортировкой по фитнесу.

# «Регуляризация»

Тоже пытается минимизировать сложность моделей (по умолчанию — отключен): рассматривает все валидные и уникальные поддеревья и выбирает среди родителей и детей лучших (эти поддеревья проще и хорошо, если они окажутся не хуже родителей)

# Inheritance(Genetic scheme)

- **generational**: новая популяция занимает место старой  $(\mu, \lambda)$ ,  $\mu = \lambda$
- **steady-state**: каждый раз добавляем по одному —  $(\mu + 1)$
- **parameter-free**:  $(\mu + x) - x$  растёт, если давно не было улучшений ( $\Rightarrow$  нужно увеличивать exploration)

# Регулятор репродукции (Population → Population)

Есть `min_size`, `max_size`; некоторые операторы вероятностны и не всегда генерируют подходящие под `GraphRequirements` графы: он пробует применять операторы несколько раз + изучает, сколько в среднем процентов успешны.

# Elitism

Поддерживать HallOfFame, во внеочередном подярке добавлять individual-ов оттуда.

# Откуда торчат уши FEDOT

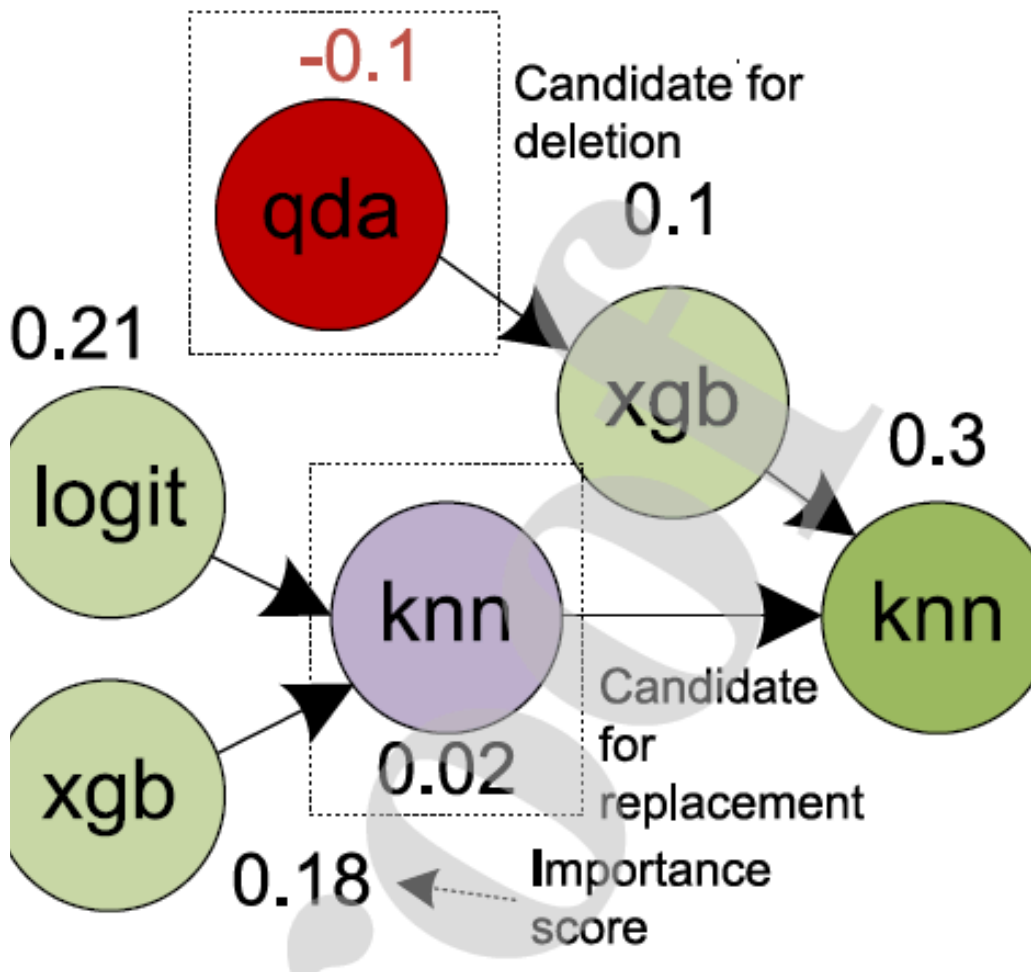
## Тюнинг

По увеличению эффективности, но уменьшению качества:

- Одновременно все ноды
- По очереди с пересчётом
- По очереди без пересчёта



# Sensitivity analysis



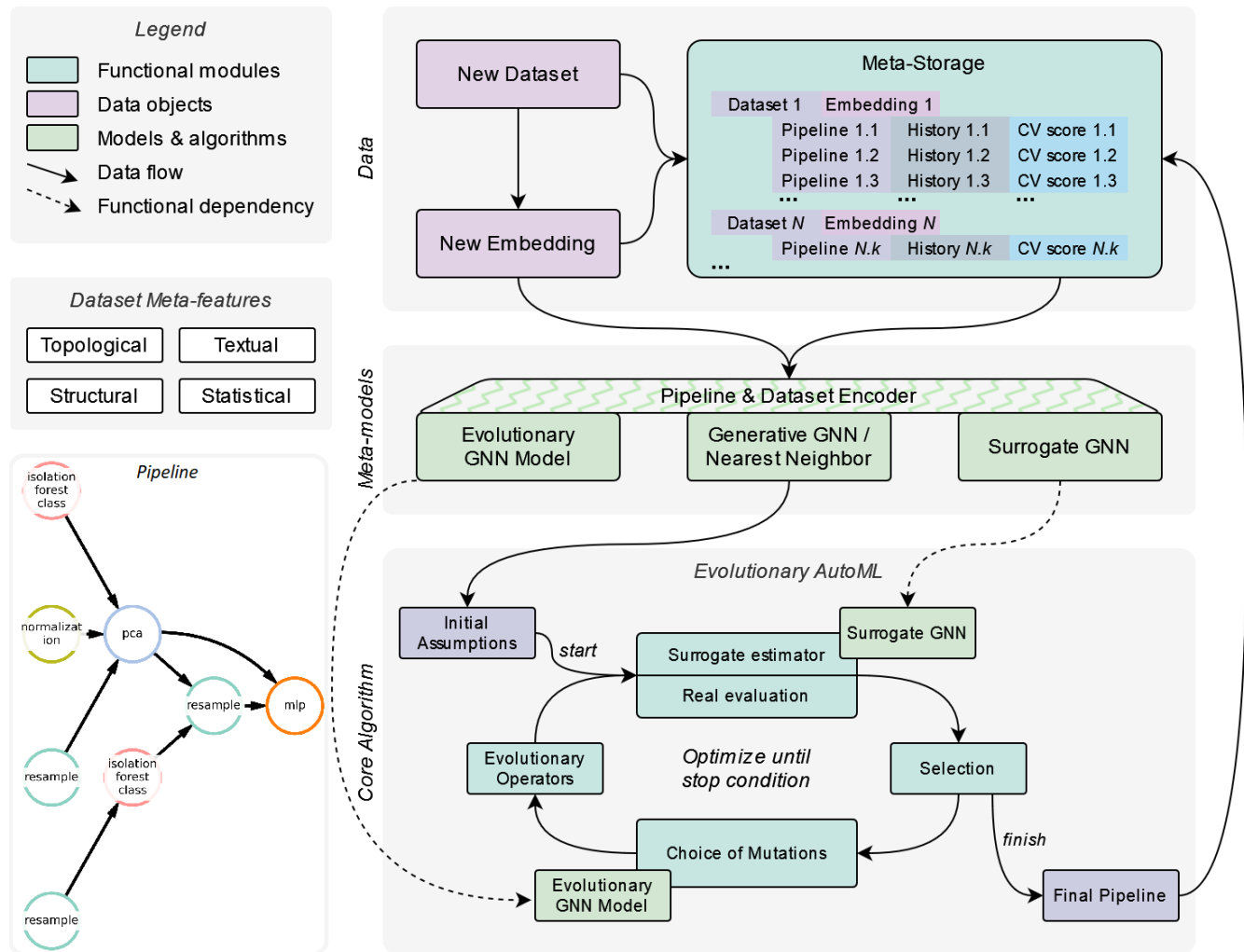
- Для экспертной оценки конечного продукта
- Назначают скоры полезности, пробуя разные локальные изменения

# Проект под кодовым названием ~~GAMLET~~ <anonymized>

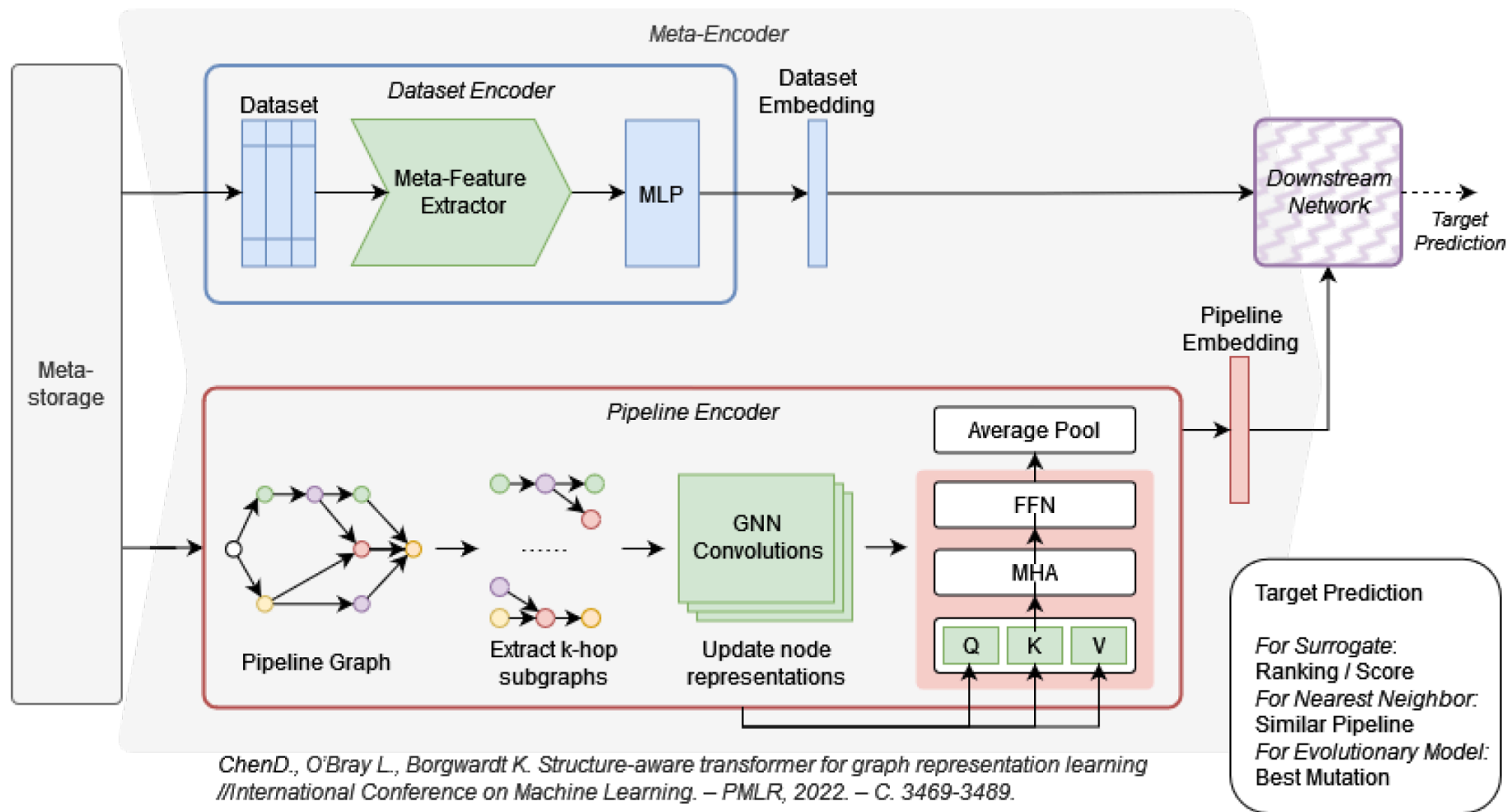
Глобально: втыкаем в разные места эволюции ML модели:

- Где мы точно не знаем, какой вариант лучше и поэтому выбираем случайный
- Где вычисление занимает очень много времени, и можно настроить аппроксиматор для его ускорения.

# Общая схема



# Расшаренный эмбединг для трёх задач:



# Адаптивность

OperatorAgent: интерфейс подборщика мутаций, по умолчанию — RandomAgent, но может обучаться. Получается задача multi-armed bandit или contextual MAB.

В качестве контекста используются фичи графов, пытаемся рекомендовать мутации для конкретного графа, обучение происходит «под конкретную задачу» в прошлых прогонах + немного на лету:

- feather\_graph ← FEATHER embedding
- nodes\_num
- labeled\_edges
- operations\_quantity
- adjacency\_matrix
- none\_encoding

## Shallow exploration & deep representation

Решают contextual MAB через нейросеть + LinUCB (сейчас пробуют  $\epsilon$ -greedy, так как он не сходится полностью, в отличие от UCB, а ведь распределение наград может меняться).

Доказывают, что это обеспечивает  $\tilde{O}(\sqrt{T})$  regret.

см. статью.

Кроме того, перед этим трансформируют в fitness improvement в credit согласно этой статье: [Li, 2014](#).

# Изначальные рекомендации pipeline-ов

Выдаёт nearest neighbors в пространстве embedding-а, либо использует генеративную GNN.

# Surrogate model

Pipeline (без учёта гиперпараметров) embedd-ится (общая часть с адаптивностью) с помощью GNN + attention (была картинка), датасет описывается метапризнаками (перечислены в статье), нужно предсказать, насколько хорошо будет себя вести модель. На практике — лучше получается формулировка с ранжированием (см. статью).



# Как это всё обучать

Есть метакранилище, где много историй оптимизации, куча evaluation-ов разных pipeline-ов на разных dataset-ах.

С големом не поставляется, с FEDOT-ом — да.

# Перспективы

Выбираются только типы мутаций, а ведь хочется ещё и их параметры (например, вероятности мутаций) — их как раз берём от балды...

Гиперпараметры эволюции всё ещё нужно указывать ручками, а ведь их можно динамически подстраивать.

Кроме того, хочется иметь более широкий контекст в виде эволюционной ситуации для бандита.

Но какой-то inductive bias всё же должен быть.

С другой стороны, есть llm-guided evolution, которая ещё и обладает априорным знанием о пайплайнах и датасетах.

# Ближе к коду

Язык: Python. Библиотеки:

- joblib + multiprocessing
- torch + mabwiser + karateclub для контекстуального бандита на GNN

*перемещаемся на демонстрацию экрана*

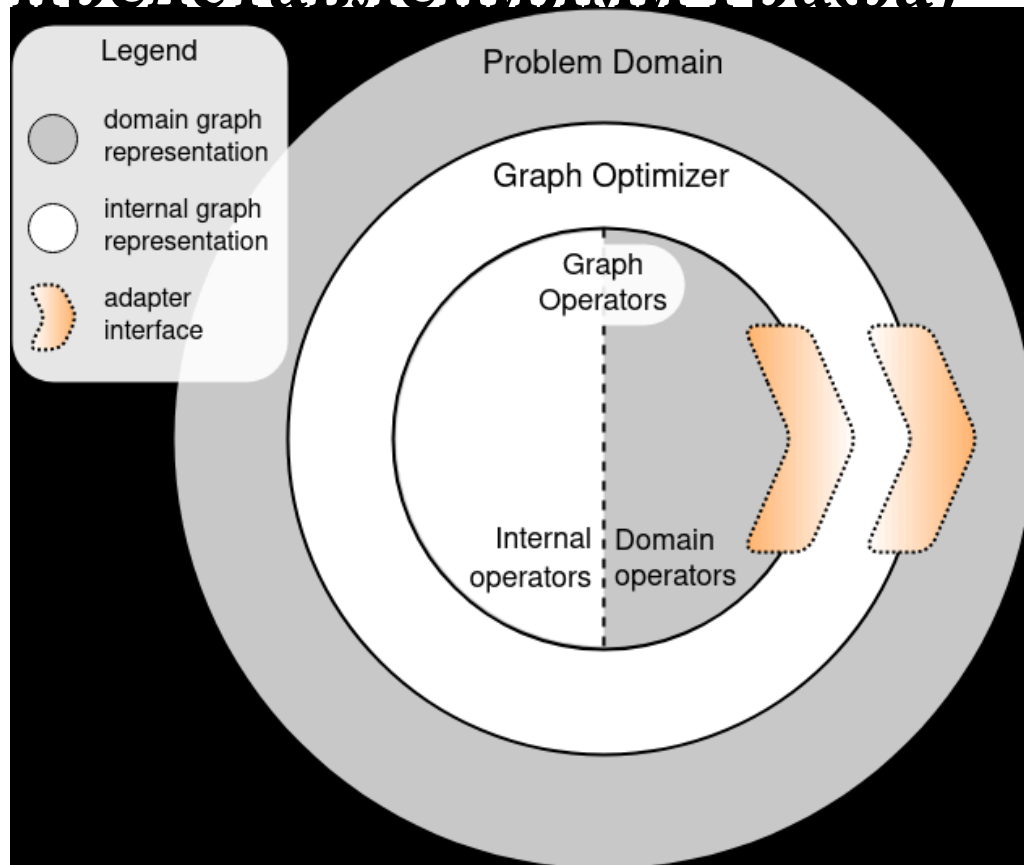
- Package core contains the main classes and scripts.
- Package core.adapter is responsible for transformation between domain graphs and internal graph representation used by optimisers.
- Package core.dag contains classes and algorithms for representation and processing of graphs.

- Package `core.optimisers` contains graph optimisers and all related classes (like those representing fitness, individuals, populations, etc.), including optimization history.
- Package `core.optimisers.genetic` contains genetic (also called evolutionary) graph optimiser and operators (mutation, selection, and so on).
- Package `core.utilities` contains utilities and data structures used by other modules.
- Package `serializers` contains class `Serializer` with required facilities, and is responsible for serialization of project classes (graphs, optimization history, and everything related).

- Package visualisation contains classes that allow to visualise optimization history, graphs, and certain plots useful for analysis.
- Package examples includes several use-cases where you can start to discover how the framework works.
- All unit and integration tests are contained in the test directory.

The sources of the documentation are in the docs directory.

# Adapter Subsystem (преобразование между представлениями графа)



- Обычно предметная область имеет своё представление графа (например, из внешней либы): химия, BAMT, FEDOT
- Fitness, операторы
- `@register_native`, e.g. `GraphVerifier`
- Поставляется адаптер к `NetworkX`
- Сейчас используется внутреннее представление в

виде рукописного графа «на  
ссылках» в python

# Сериализация

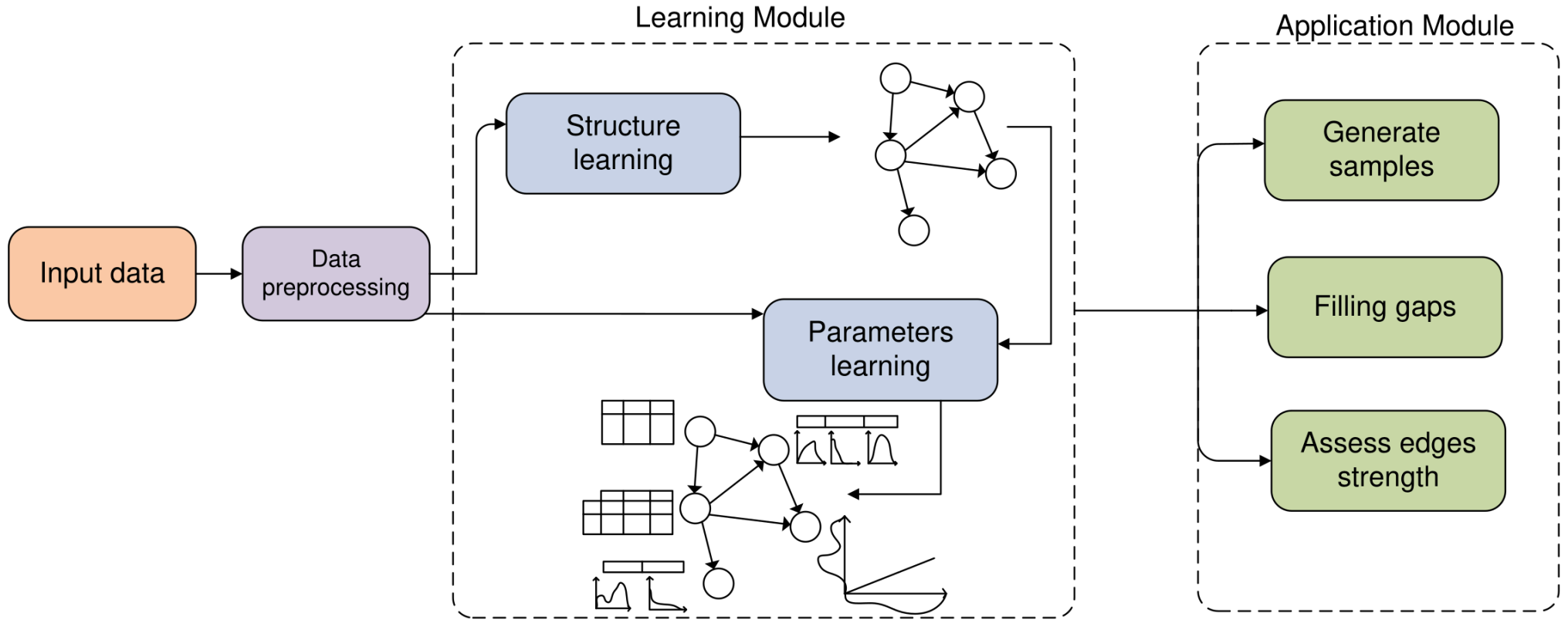
- Pickle (e.g. бандиты)
- json (в т.ч. pipeline-ы в FEDOT)



# Issues

*перемещаемся на демонстрацию экрана*

# Пример использования: ВАМТ



Есть вещественные и непрерывные переменные, от этого зависит процесс обучения. Описано тут: [Advanced Approach for Distributions Parameters Learning in Bayesian Networks with Gaussian Mixture Models and Discriminative Models \(2023\)](#)

Тюняются в порядке topsort-а DAG-а, для зависимости от дискретной используется CPT, для зависимости непрерывных — линейные, гауссовы зависимости, Gaussian mixture regression.

# Направления развития

Источники:

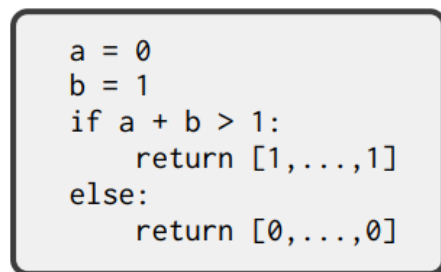
- Issues
- Разговоры с коллективом лаборатории
- Мои рассуждения
- Рекомендации из review к paper по GOLEM-у.

# Expressive encodings

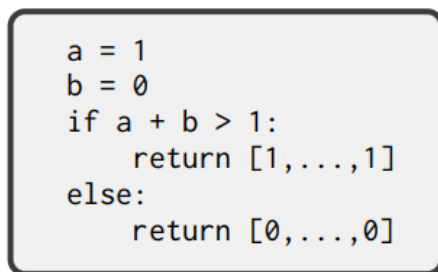
- Прямо как в жизни — в геноме кодируется распределение, не просто какой-то конкретный фенотип, а больше информации. Заодно — и, частично, то, как проходит эволюция.
- Приближает к положению дел в ML, где используют достаточно абстрактные модели вместо экспертного знания.

**Definition 1** (Expressive Encoding). An encoding  $E : X \rightarrow Y$  is *expressive* for a simple genetic operator  $g$  if, for any set of parent phenotypes  $\{y_1, \dots, y_{n_g}\} = Y_p \subset Y$ , any probability density  $\mu$  over  $Y$ , and any  $\epsilon > 0$ , there exists a set of parent genotypes  $\{x_1, \dots, x_{n_g}\} = X_p \subset X$  such that  $E(x_i) = y_i \ \forall y_i \in Y_p$ , and

$$\left| \Pr [E(g(X_p)) = y] - \mu(y) \right| < \epsilon \ \forall y \in Y. \quad (2)$$

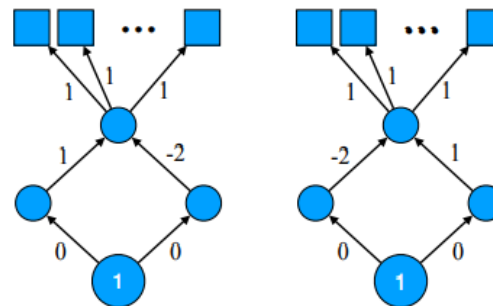


Parent 1



Parent 2

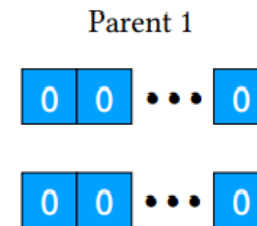
(a)



Parent 1

Parent 2

(b)



Parent 2

(c)

**Figure 1: *Miracle Jump Parents*.** (a) Two GP parents whose phenotypes are all 0's, but whose crossover has maximal jump (to a child of all 1's) with probability 0.25. They differ only in their values of  $a$  and  $b$ ; the probability is independent of phenotype dimensionality. (b) Two NN parents with this same property; They differ only in the weights in the second layer. (c) Directly encoded parents cannot have this property: If both parents are all 0's, their crossover cannot yield all 1's. This minimal example illustrates how expressive encodings yield high-dimensional structured behavior that direct encodings cannot capture.

# Примеры expressive encodings

Complexity — размер генома, требуемый для фиксированного уровня аппроксимации.

**THEOREM 4.2.** *Genetic programming is an expressive encoding for single-point mutation, with complexity  $O(\frac{mn}{\epsilon})$ .*

**THEOREM 4.4.** *Feed-forward neural networks with sigmoid activation are an expressive encoding for single-point mutation, with complexity  $O(\frac{mn}{\epsilon})$ .*

**Definition 2** ( $E_\Omega$ ). Let  $\Omega$  be any universal function approximator. Define  $E_\Omega$  to be an encoding whose genotypes are of the form  $\omega(\mathbf{a})$ , where  $\mathbf{a} \in \{0, 1\}^L$ , and  $\omega \in \Omega$  is a function  $\omega : \{0, 1\}^L \rightarrow Y$ .

**THEOREM 4.5.**  *$E_\Omega$  is an expressive encoding for uniform crossover.*

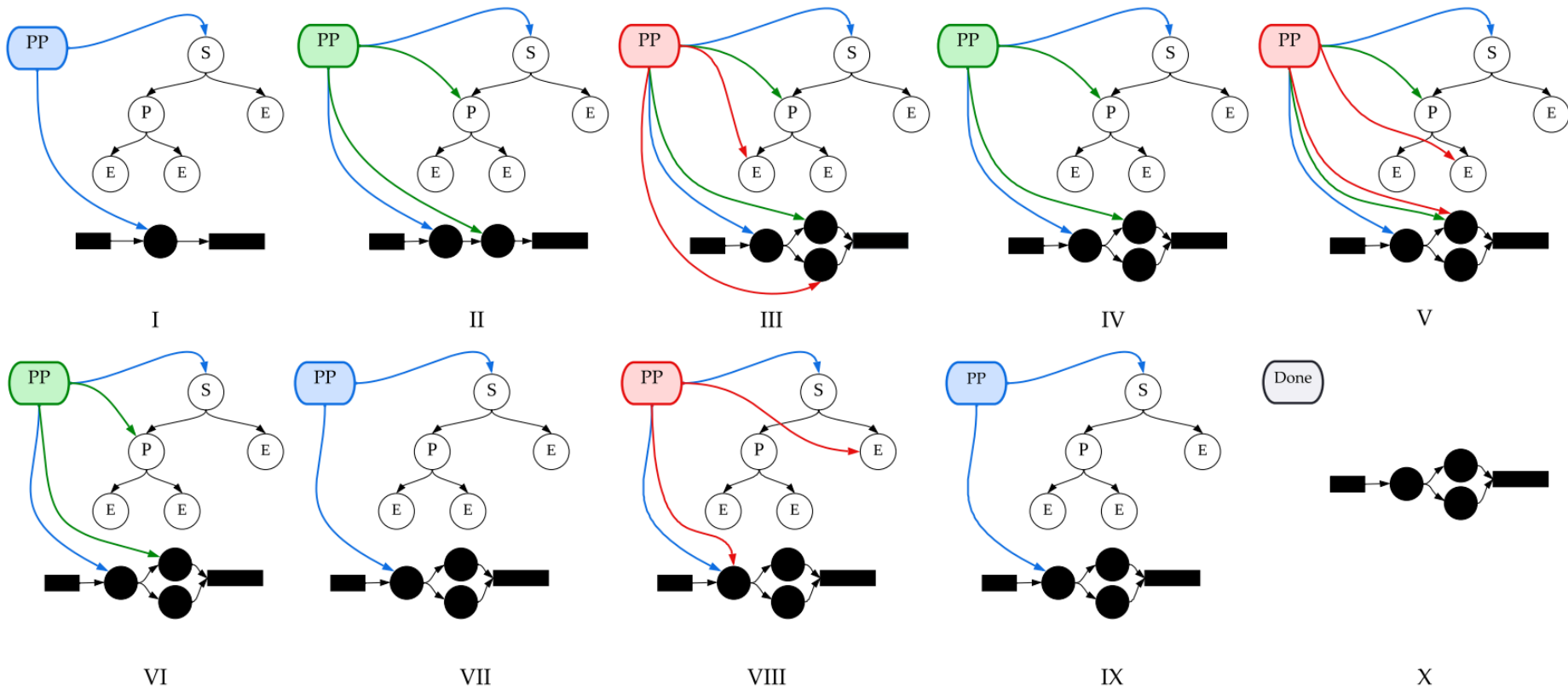


Потом на разных примерах показывают более хорошую асимптотику expressive encoding-ов по сравнению с direct.

В нашем случае кодировкой может быть

- графовые грамматики
- клеточное кодирование
- GNN

*с тривиальными мутациями/кроссоверами.*



**Fig. 3.** Growing a neural network from an ancestor network. **PP**=Program Pointers, **S**=Sequential Split, **P**=Parallel Split, **E**=End Terminal.

# NOTES

**Theorem 1.** *A matrix  $W \in \mathbb{R}^{d \times d}$  is a DAG if and only if*

$$h(W) = \text{tr}(e^{W \circ W}) - d = 0, \quad (7)$$

*where  $\circ$  is the Hadamard product and  $e^A$  is the matrix exponential of  $A$ . Moreover,  $h(W)$  has a simple gradient*

$$\nabla h(W) = (e^{W \circ W})^T \circ 2W, \quad (8)$$

*and satisfies all of the desiderata (a)-(d).*

→ сводим к задаче непрерывной оптимизации; целевая функция и ограничения — дифференцируемы. Решаем с помощью метода расширенной Лагранжианы.

Ограничения:

- Целевая функция должна естественно (дифференцируемо, поменьше константа Липшица, легко вычислима) продолжаться на вещественные веса
- Как задать на пространстве, содержащем категориальные переменные?

Как может совмещаться с GOLEM-ом? Идея: оператор локального улучшения.

# Метаэволюция

Пытаемся адаптировать гиперпараметры эволюции (частоту мутаций, размер популяции и т.д.). Похоже на предложение к GAMLET, но теперь эти параметры именно эволюционируют.

см. Parameter Control in Evolutionary Algorithms: Trends and Challenges

# Коэволюция

см. proposal.

# Поддержка разнообразия

см. proposal.