Дата последней сборки документа:
12 ноября 2021 г. в 05:09

Аналитическая справка по стратегиям применения подходов машинного/глубокого обучения в задачах частично-целочисленного линейного программирования

Содержание

1. Общие замечания

Во многих промышленных приложениях таких как планирование производства и оптимизация цепочки поставок часто приходится решать NP-трудные задачи комбинаторной оптимизации. На практике такого рода задачи решаются с помощью солверов общего назначения, которые обрабатывают каждый отдельный расчетный случай независимо и не обобщают накопленный опыт решения аналогичных задач. Тем неменее, весьма вероятно, что между этими задачами существует сильные статистические зависимости, которые можно было бы использовать.

В работе [1] отмечается, что на текущий момент выделяется два направления, ориентированных на то, чтобы информация о расчетных прецедентах использовалась в ходе решения:

- подход на базе «чистого» машинного обучения, который подменяет решатели задач комбинаторной оптимизации и направлен на создание около-оптимальных решений [2], [3], [4]
- и подход, в котором разработанные вручную критерии принятия решений в рамках классических решателей задач комбинаторной оптимизации заменяются моделями машинного обучения, настроенными на оптимизацию конкретной метрики решателя. Этот подход по мнению авторов работы [1] представляется наиболее перспективным, поскольку он позволяет получить *точное решение* или, по крайней мере, *математически обоснованные* границы оптимальной области, которые часто имеют большое значение на практике [6].

Ниже рассматриваются наиболее популярные инструменты и наиболее перспективные стратегии использования аппарата машинного/глубокого обучения применительно к решению задач комбинаторной оптимизации в постановке частично-целочисленного линейного программирования (Mixed Integer Linear Program, MILP).

1.1. Утилита командной строки scip

Командный инструмент scip https://www.scipopt.org/ предоставляет консольный интерфейс для конфигурирования и управления решателем SCIP.

SCIP – это один самых быстрых некоммерческих решателей для задач комбинаторной оптимизации в частично-целочисленной постановке линейного (MILP) и нелинейного программирования (MINLP).

В SCIP реализовано большое число эвристик (как самых простых, так и state-of-the-art)

Результат работы команды set/heuristics

```
<actconsdiving>       LP diving heuristic that chooses fixings w.r.t. the active constraints
<adaptivediving>      diving heuristic that selects adaptively between the existing, public
    divesets
<advanced>            advanced parameters
<alns>                Large neighborhood search heuristic that orchestrates the popular
    neighborhoods Local Branching, RINS, RENS, DINS etc.
<bound>               heuristic which fixes all integer variables to a bound and solves the
    remaining LP
<clique>              LNS heuristic using a clique partition to restrict the search neighborhood
<coefdiving>          LP diving heuristic that chooses fixings w.r.t. the matrix coefficients
<completesol>         primal heuristic trying to complete given partial solutions
<conflictdiving>      LP diving heuristic that chooses fixings w.r.t. conflict locks
<crossover>           LNS heuristic that fixes all variables that are identic in a couple of
    solutions
<dins>                distance induced neighborhood search by Ghosh
<distributiondiving>  Diving heuristic that chooses fixings w.r.t. changes in the solution
    density
<dualval>             primal heuristic using dual values
<emphasis>            predefined parameter settings
<farkasdiving>        LP diving heuristic that tries to construct a Farkas-proof
<feaspump>            objective feasibility pump 2.0
<fixandinfer>         iteratively fixes variables and propagates inferences
<fracdiving>          LP diving heuristic that chooses fixings w.r.t. the fractionalities
<gins>                gins works on k-neighborhood in a variable-constraint graph
<guideddiving>        LP diving heuristic that chooses fixings in direction of incumbent
    solutions
...
```

```
 <proximity>           heuristic trying to improve the incumbent by an auxiliary proximity
    objective function
<pscostdiving>        LP diving heuristic that chooses fixings w.r.t. the pseudo cost values
<randrounding>        fast LP rounding heuristic
<rens>                LNS exploring fractional neighborhood of relaxation's optimum
<reoptsols>           primal heuristic updating solutions found in a previous optimization round
<repair>              tries to repair a primal infeasible solution
<rins>                relaxation induced neighborhood search by Danna, Rothberg, and Le Pape
<rootsoldiving>       LP diving heuristic that changes variable's objective values using root LP
     solution as guide
<rounding>            LP rounding heuristic with infeasibility recovering
<shiftandpropagate>   Pre-root heuristic to expand an auxiliary branch-and-bound tree and apply
    propagation techniques
<shifting>            LP rounding heuristic with infeasibility recovering also using continuous
    variables
<simplerounding>      simple and fast LP rounding heuristic
...
```

1.2. Библиотека-интерфейс PyScipOpt к решателю SCIP

PyScipOpt https://github.com/scipopt/PySCIPOpt предоставляет оболочку над решателем SCIP.

Установить библиотеку можно с помощью менджера пакетов `conda`

```
conda install --channel conda-forge pyscipopt
```

Пример использования

```
import pyscipopt

SCIP_PARAMS = {
    "display/lpinfo": True,
    "limits/gap": 10,
    "limits/solutions": 1
    "presolving/maxrounds": 0,
    "presolving/maxrestarts": 0
}

model = pyscipopt.Model("./file.lp")
model.setParams(SCIP_PARAMS)
model.solve()
```

2. Библиотека OR-Tools и платформа Google Optimization Tools

2.1. Установка

Установить пакет можно следующим образом

```
pip install ortools
```

2.2. Общие сведения

Google Optimization Tools (OR-Tools) https://github.com/google/or-tools – быстрый, портативный пакет с открытым исходным кодом для решения задач комбинаторной оптимизации.

Пример использования

```
from ortools.linear_solver import pywraplp
from ortools.init import pywrapinit


def main():
  # Create the linear solver with the GLOP backend.
  solver = pywraplp.Solver.CreateSolver('GLOP')

  # Create the variables x and y.
  x = solver.NumVar(0, 1, 'x')
  y = solver.NumVar(0, 2, 'y')

  print('Number of variables =', solver.NumVariables())

  # Create a linear constraint, 0 <= x + y <= 2.
  ct = solver.Constraint(0, 2, 'ct')
  ct.SetCoefficient(x, 1)
  ct.SetCoefficient(y, 1)

  print('Number of constraints =', solver.NumConstraints())

  # Create the objective function, 3 * x + y.
  objective = solver.Objective()
  objective.SetCoefficient(x, 3)
  objective.SetCoefficient(y, 1)
  objective.SetMaximization()

  solver.Solve()

  print('Solution:')
  print('Objective value =', objective.Value())
  print('x =', x.solution_value())
  print('y =', y.solution_value())


if __name__ == '__main__':
  pywrapinit.CppBridge.InitLogging('basic_example.py')
  cpp_flags = pywrapinit.CppFlags()
  cpp_flags.logtostderr = True
  cpp_flags.log_prefix = False
  pywrapinit.CppBridge.SetFlags(cpp_flags)

  main()
```

2.3. Достоинства и недостатки

(+) Достоинства:

- Библиотека с открытым исходным кодом,
- Поддерживается всеми популярными операционными системами,
- Простой, дружелюбный интерфейс,
- Есть поддержка частично-целочисленной постановки линейного программирования.

(-) Недостатки:

- Не поддерживает lp-файлы. Это значит, что существующими lp-файлами, подготовленными с помощью MIP-Python воспользоваться не удастся и придется переписывать весь проект на базе библиотеки `ortools`.

3. Нейронные сети как стратегия

3.1. Графовые и бинаризованные нейронные сети

К ключевым публикациям, затрагивающим вопросы использования *графовых нейронных сетей* (Graph Neural Network, GNN) в контексте задач комбинаторной оптимизации (MILP-постановка), можно отнести работы Gupta etc. [5], Bengio etc. [6] и Gasse M. etc. [7].

Резльтаты их исследований показывают, что подход, основанный на сверточных графовых нейронных сетях применительно к некоторым классам задач комбинаторной оптимизации, может значительно снизить временные издержки на получение решения. Однако, графовые нейронные сети для эффективной работы требуют кластера машин с графическим процессором, что с практической точки зрения представляется недостатком подхода.

В работе [10] предлагается MIP-переменные x^l моделировать как выходной вектор l-слоя глубокой нейронной сети ($l > 0$ и l^0 – входной вектор)

$$x^l = ReLU(W^{l-1}x^{l-1} + b^{l-1}), \quad \forall l = 1, \dots, L.$$

Тогда постановка для смешанной целочисленной линейной задачи будет выглядеть следующим образом

$$\begin{aligned}
&\min \sum_{l=0}^{L} \sum_{j=1}^{n_l} c_j^l x_j^l + \sum_{l=1}^{L} \sum_{j=1}^{n_l} \gamma_j^l z_j^l, \\
&\sum_{i=1}^{n_l-1} w_{ij}^{l-1} x_i^{l-1} + b_j^{l-1} = x_j^l - s_j^l, \quad \forall l = 1, \dots, L, \ j = 1, \dots, n_l, \\
&x_j^l \leqslant (1 - z_j^l) M_x^{j,l}, \quad \forall l = 1, \dots, L, \ j = 1, \dots, n_l, \\
&s_j^l \geqslant z_j^l M_s^{j,l}, \quad \forall l = 1, \dots, L, \ j = 1, \dots, n_l, \\
&0 \leqslant x_j^l \leqslant ub_j^l, \quad \forall l = 1, \dots, L, \ j = 1, \dots, n_l, \\
&0 \leqslant s_j^l \leqslant ub_j^l, \quad \forall l = 1, \dots, L, \ j = 1, \dots, n_l,
\end{aligned}$$

где $M_x^{j,l}$, $M_s^{j,l}$ – некоторые на перед заданные большие константы.

В работе [11] для решения MIP-задач предлагается использовать бинаризованные нейронные сети (Binarized Neural Networks, BNNs). Бинаризованные нейронные сети описываются бинарными весами $\{-1, +1\}$ и используют функцию знаков для активации нейронов.

Как правило, BNNs обнаруживают более высокую обобщающую способность (особенно для небольших наборов данных).

3.2. Комбинированные архитектуры нейронных сетей

Для преодоления трудностей, связанных с высокими накладными расходами как в части инфраструктуры модели, так и в части времени расчета, в работе Gupta [5] была предложена *гибридная архитектура*, направленная на эффективное решение задачи ветвления в branch-and-bound деревьях.

Замечание

Авторы работы [5] подчеркивают, что «классические» графовые нейронные сети могут быть эффективны только на кластере, поддерживающем графические процессоры

Эта архитектура с одной стороны учитывает выразительную мощность графовых нейронных сетей, а с другой – вычислительную эффективность многослойного персептрона, включающегося в цепочку вычислений на этапе ветвления дерева.

По заявлениям авторов гибридная модель (на 4 наборах данных) сокращает временны*е* издержки до 26% (используется центральный процессор, не графический!).

3.3. Достоинства и недостатки

(+) Достоинства:

- Нейронные сети предлагают *гибкий* инструмент управления процедурой решения задач комбинаторной оптимизации в частично-целочисленной постановке,

(-) Недостатки:

- Мало ресурсов, проясняющих тонкости программной реализации (по большей части встречаются только теоретические работы).

4. Библиотека Ecole

4.1. Установка

На Unix-подобные операционные системы[1] библиотеку можно установить с помощью менеджера пакетов `conda`

```
conda install -c conda-forge ecole
```

4.2. Общие сведения

Библиотека Ecole https://www.ecole.ai/ разработана специально для высокоуровневого управления решателем SCIP https://www.scipopt.org/.

Цель Ecole – предоставить абстракции марковского процесса принятия решений в отношении задач комбинаторной оптимизации. Эти задачи представлены классами с отслеживаемым состоянием, которые называются средами/окружениями (environments).

Ядро Ecole написано на C++, напрямую взаимодействует с API решателя и предоставляет тонкий Python-интерфейс, возвращающий массивы NumPy для взаимодействия с ML-библиотеками.

На текущий момент поддерживается только решатель SCIP, но в планах разработчиков сделать возможным свободную поддержку и коммерческих решателей (Gurobi, CPLEX, XPress etc.).

В настоящее время Ecole поддерживает две *задачи управления* (control tasks):

- подбор гиперпараметров решателя на этапе предобработки (`ecole.enviroment.Configuring`),
- отбор переменных (`ecole.enviroment.Branching`); на каждой итерации построения branch-and-bounds дерева принимается решение о выборе следующей переменной для ветвления.

Библиотека поддерживает две *функции наблюдения* (observation functions) за состоянием процесса решения:

[1] На текущий момент поддерживается *только* Unix-подобные операционные системы

- конечномерное агрегированное представление переменных по Khalil,
- двудольное графическое представление по Gasse [7].

Также билиотека поддерживает две стандартные *функции вознаграждения* (reward functions), а именно:

- количество узлов в branch-and-bound дереве,
- количество итераций LP, добавленных с момента последнего принятного решения.

Теоретическая модель Ecole описана на странице проекта https://doc.ecole.ai/py/en/stable/discussion/theory.html.

С сигнаторой классов для функций наблюдения, вознаграждения и пр. элементами библиотеки можно ознакомится на странице документации https://doc.ecole.ai/py/en/stable/.

4.3. Достоинства и недостатки

(+) Достоинства:

- Библиотека с открытым исходным кодом,
- Дружелюбный интерфейс,
- Есть поддержка частично-целочисленной постановки линейного программирования,
- Реализованы различные эффективные сценарии управления процедурой решения задач комбинаторной оптимизации.

(-) Недостатки:

- Слабая документация,
- Библиотека поддерживается только на Unix-подобных операционных системах,
- Очень мало ресурсов проясняющих тонкие моменты работы с библиотекой.

5. Выводы и рекомендации

Обобщив сказанное выше, можно выделить две ключевые стратегии применения аппарата машинного/глубокого обучения для MILP:

- Стратегия на нейронных сетях. «Чистые» графовые сверточные нейронные сети обнаруживают возможности для снижения временны*х* издержек на задачах комбинаторной оптимизации в частично-целочисленной постановке, но требуют специализированной дорогостоящей инфраструктуры. А вопрос применения бинаризованных нейронных сетей на данный момент следует считать дискуссионным.
- Стратегия оборачивания «классических» решателей логикой алгоритмов машинного/глубокого обучения для редуцирования размероности задачи по переменным в branch-and-bound дереве и подбора гиперпараметров решателя.

Вторая стратегия представляется наиболее реалистичной. На текущий момент существует только одна библиотека, которая работает напрямую с решателем SCIP (на низком уровне), реализует наиболее эффективные алгоритмы машинного обучения и не требует специально сконфигурированной инфраструктуры[2] – это библиотека Ecole.

Рекомендуется более глубоко исследовать возможности библиотеки Ecole как кандидата на роль *базового каркаса динамического прототипа* для модульно-расширяемых решений.

Тогда типовая схема использования Ecole будет выглядеть следующим образом:

[2]Решение в самом простом случае может работать и только на центральном процессере

- (отдельный Python-модуль) Прочитать входной json-файл и провести валидацию,
- (отдельный Python-модуль) На основании входного json-файла подготовить lp-файл, описывающий математическую постановку задачи (может использоваться любая обертка над любым решателем с открым исходным кодом: MIP-Python для CBC-решателя, PyScipOpt для SCIP-решателя и т.д.),
- (отдельный Python-модуль) Сконфигурировать окружение агента и начать обучение,
- Сохранить полученное не предыдущем шаге решение как sol-файл,
- (отдельный Python-модуль) Отобразить sol-файл на выходной json-файл.

Основные логические блоки изолированы друг от друга и взаимодействуют через тонкий интерфейс. В случае необходимости расширить функционал решения в какой-либо части, например в части новой логики алгоритмов машинного обучения, будет достаточно внести изменения лишь в один соответсвующий модуль, не затрагивая остальные.

A. Пример реализации графовой сверточной нейронной сети из работы Gasse [7] с помощью библиотеки Ecole

```
import sys
import torch
import torch_geometric
import logging
import ecole

SCIP_PARAMS = {
  "display/lpinfo": True,
  "limits/time": 1200,
  "separating/maxrounds": 0,
  "presolving/maxrestarts": 0,
}

INPUT_LP_FILE = "./planner_from_MIP_wo_min_and_int.lp"
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")


class GNNPolicy(torch.nn.Module):
  def __init__(self):
    super().__init__()
    emb_size = 64
    cons_nfeats = 5
    edge_nfeats = 1
    var_nfeats = 19

    # CONSTRAINT EMBEDDING
    self.cons_embedding = torch.nn.Sequential(
      torch.nn.LayerNorm(cons_nfeats),
      torch.nn.Linear(cons_nfeats, emb_size),
      torch.nn.ReLU(),
      torch.nn.Linear(emb_size, emb_size),
      torch.nn.ReLU(),
    )

    # EDGE EMBEDDING
    self.edge_embedding = torch.nn.Sequential(torch.nn.LayerNorm(edge_nfeats),)

    # VARIABLE EMBEDDING
```

```
    self.var_embedding = torch.nn.Sequential(
      torch.nn.LayerNorm(var_nfeats),
      torch.nn.Linear(var_nfeats, emb_size),
      torch.nn.ReLU(),
      torch.nn.Linear(emb_size, emb_size),
      torch.nn.ReLU(),
    )

    self.conv_v_to_c = BipartiteGraphConvolution()
    self.conv_c_to_v = BipartiteGraphConvolution()

    self.output_module = torch.nn.Sequential(
      torch.nn.Linear(emb_size, emb_size),
      torch.nn.ReLU(),
      torch.nn.Linear(emb_size, 1, bias=False),
    )

  def forward(
    self, constraint_features, edge_indices, edge_features, variable_features
  ):
    reversed_edge_indices = torch.stack([edge_indices[1], edge_indices[0]], dim=0)

    # First step: linear embedding layers to a common dimension (64)
    constraint_features = self.cons_embedding(constraint_features)
    edge_features = self.edge_embedding(edge_features)
    variable_features = self.var_embedding(variable_features)

    # Two half convolutions
    constraint_features = self.conv_v_to_c(
      variable_features, reversed_edge_indices, edge_features, constraint_features
    )
    variable_features = self.conv_c_to_v(
      constraint_features, edge_indices, edge_features, variable_features
    )

    # A final MLP on the variable features
    output = self.output_module(variable_features).squeeze(-1)
    return output


class BipartiteGraphConvolution(torch_geometric.nn.MessagePassing):
  """
  The bipartite graph convolution is already provided by pytorch geometric and we merely need
  to provide the exact form of the messages being passed.
  """

  def __init__(self):
    super().__init__("add")
    emb_size = 64

    self.feature_module_left = torch.nn.Sequential(
      torch.nn.Linear(emb_size, emb_size)
    )
    self.feature_module_edge = torch.nn.Sequential(
      torch.nn.Linear(1, emb_size, bias=False)
    )
    self.feature_module_right = torch.nn.Sequential(
      torch.nn.Linear(emb_size, emb_size, bias=False)
    )
    self.feature_module_final = torch.nn.Sequential(
```

```
      torch.nn.LayerNorm(emb_size),
      torch.nn.ReLU(),
      torch.nn.Linear(emb_size, emb_size),
    )

    self.post_conv_module = torch.nn.Sequential(torch.nn.LayerNorm(emb_size))

    # output_layers
    self.output_module = torch.nn.Sequential(
      torch.nn.Linear(2 * emb_size, emb_size),
      torch.nn.ReLU(),
      torch.nn.Linear(emb_size, emb_size),
    )

  def forward(self, left_features, edge_indices, edge_features, right_features):
    """
    This method sends the messages, computed in the message method.
    """
    output = self.propagate(
      edge_indices,
      size=(left_features.shape[0], right_features.shape[0]),
      node_features=(left_features, right_features),
      edge_features=edge_features,
    )
    return self.output_module(
      torch.cat([self.post_conv_module(output), right_features], dim=-1)
    )

  def message(self, node_features_i, node_features_j, edge_features):
    output = self.feature_module_final(
      self.feature_module_left(node_features_i)
      + self.feature_module_edge(edge_features)
      + self.feature_module_right(node_features_j)
    )
    return output


policy = GNNPolicy().to(DEVICE)


def optimize_model():
  env = ecole.environment.Branching(
    scip_params=SCIP_PARAMS,
    observation_function=ecole.observation.MilpBipartite(),
    reward_function=ecole.reward.NNodes(),
    information_function={
      "nb_nodes": ecole.reward.NNodes().cumsum(),
      "time": ecole.reward.SolvingTime().cumsum(),
    },
  )

  logger.info("Reset environment ...")
  nb_nodes, time = 0, 0
  (obs, action_set, reward, done, info) = env.reset(INPUT_LP_FILE)
  nb_nodes += info["time"]

  while not done:
  logger.info("New step in environment ...")
  with torch.no_grad():
  obs = (
```

```
    torch.from_numpy(obs.row_features.astype(np.float32)).to(DEVICE),
    torch.from_numpy(obs.edge_features.indices.astype(np.int64)).to(DEVICE),
    torch.from_numpy(obs.edge_feature.values.astype(np.float32))
    .view(-1, 1)
    .to(DEVICE),
    torch.from_numpy(obs.column_features.astype(np.float32)).to(DEVICE),
  )
  logits = policy(*obs)
  action = action_set[logits[action_set.astype(np.int64)].argmax()]
  (obs, action_set, reward, done, info) = env.step(action)
  nb_nodes += info["nb_nodes"]
  time += info["time"]

  return env.model.as_pyscipopt()


if __name__ == "__main__":
_LOG_FORMAT = "{asctime}: {levelname} -> {message}"

  stream_handler = logging.StreamHandler(sys.stdout)
  stream_handler.setFormatter(logging.Formatter(_LOG_FORMAT, style="{"))

  logger = logging.getLogger(__name__)
  logger.setLevel(logging.INFO)
  logger.addHandler(stream_handler)

  logger.info("Starting compute ...")
  model = optimize_model()

  logger.info("Writing statistic information ...")
  model.writeStatistics(filename="planner_stat_from_ecole.stats")

  logger.info("Writing best solution information ...")
  model.writeBestSol(filename="planner_sol_from_ecole.sol")
```

Список литературы

1. *Prouvost, A. etc.* Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers, 2020 https://arxiv.org/pdf/2011.06069v2.pdf
2. *Bello, I. etc.* Neural combinatorial optimization with reinforcement learning. In Proceedings of the Fifth International Conference of Learning Representations, 2017
3. *Dai, H. etc.* Learning combinatorial optimization algorithms over graphs. In Advances in Neural Information Processing Systems, 2017
4. *Kool, W. etc.* Attention, learn to solve routing problems! International Conference on Learning Representations, 2019
5. *Gupta, P. etc.* Hybrid Models for Learning to Branch, 2020 https://paperswithcode.com/paper/hybrid-models-for-learning-to-branch
6. *Bengio, Y. etc.* Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon https://arxiv.org/abs/1811.06128v2
7. *Gasse M. etc.* Exact Combinatorial Optimization with Graph Convolutional Neural Networks https://arxiv.org/abs/1906.01629
8. *Khalil, E. etc.* Learning to branch in mixed ineteger programming. In Dale Schuurmans and Michael P. Wellman, AAAI, pages 724-731. AAAI Press, 2016

9. *Gambella, C. etc.* Optimization Problem for Maching Learning: A Survey https://arxiv.org/pdf/1901.05331.pdf
10. *Fischetti, M., Jason, Jo.* Deep neural networks and mixed integer linear optimization. Constraints, 23(3): 296-309, 2018
11. *Khalil, E. etc.* Combinatorial attacks on binarized neural networks. Technical report, arXiv preprint 1810.03538, 2018