ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 36 за 2022 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

И.А. Белозеров, В.А. Судаков

Машинное обучение с подкреплением для решения задач математического программирования

Статья доступна по лицензии
Creative Commons Attribution 4.0 International

Рекомендуемая форма библиографической ссылки: Белозеров И.А., Судаков В.А. Машинное обучение с подкреплением для решения задач математического программирования // Препринты ИПМ им. М.В.Келдыша. 2022. № 36. 14 с. https://doi.org/10.20948/prepr-2022-36
https://library.keldysh.ru/preprint.asp?id=2022-36

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

И.А. Белозеров, В.А. Судаков

Машинное обучение с подкреплением для решения задач математического программирования

Москва — 2022

Белозеров И. А., Судаков В. А.

Машинное обучение с подкреплением для решения задач математического программирования.

В работе рассматриваются современные подходы к поиску рациональных решений в задачах смешанного целочисленного линейного программирования, как сгенерированных со случайными данными, так и из реальной практики. Основной упор сделан на то, каким образом можно осуществить процесс поиска решения задач дискретной оптимизации, используя концепцию обучения с подкреплением; какие техники возможно применить, чтобы улучшить скорость и качество работы. Были разработаны три основных варианта алгоритма, с помощью API библиотеки Ray, а также окружения – библиотеки Gym. Приводится сравнение результатов разработанного решателя с библиотекой OR-Tools. Лучшая модель может быть использована в качестве решателя для оптимизационных задач большой размерности, кроме того, данная концепция применима к другим задачам комбинаторного характера с изменением кода окружения и алгоритма интеллектуального агента.

Ключевые слова: машинное обучение с подкреплением, окружение, нейронные сети, смешанное целочисленное программирование, дискретная оптимизация Ray, Gym.

Ilya Andreevich Belozerov, Vladimir Anatolyevich Sudakov.

Reinforcement Machine Learning for Solving Mathematical Programming Problems.

This paper discusses modern approaches to finding rational solutions in problems of mixed integer linear programming, both generated with random data and from real practice. The main emphasis is on how to implement the process of finding a solution to discrete optimization problems using the concept of reinforcement learning; what techniques can be applied to improve the speed and quality of work. Three main variants of the algorithm were developed using the Ray library API, as well as the environment - the Gym library. The results of the developed solver are compared with the OR-Tools library. The best model can be used as a solver for high-dimensional optimization problems, in addition, this concept is applicable to other combinatorial problems with a change in the environment code and the intelligent agent algorithm.

Key words: reinforcement learning, environment, neural networks, mixed integer programming, discrete optimization, Ray, Gym.

Работа выполнена при финансовой поддержке ФГБОУ ВО "РЭУ им. Г.В. Плеханова".

Введение

Большое количество задач в области планирования, рационального использования ресурсов, поддержки и принятии решений [4] в экономике, логистике, на производстве товаров и услуг сводятся к задачам линейного, дискретного и смешанного целочисленного программирования. При большом количестве переменных и ограничений в подобных задачах становится невозможным найти решения в силу их NP-трудности. Существование суперкомпьютерных технологий также не сможет обеспечить гарантированного нахождения решений за требуемое время, так как в основном данные вычислительные мощности эффективнее использовать для задач моделирования физических процессов либо для машинного и глубинного обучения.

В данной работе исследуются способы и методы решений задач оптимизации комбинаторного характера методами машинного обучения с подкреплением, также приводится сравнение разработанных подходов на задачах MIP соответствующих размерностях с методом ветвей и границ, реализованных в библиотеке Google – or-tools. Кроме того, рассматривается применимость современных алгоритмов обучения с подкреплением, таких как: A3C, DQN, PPO, а именно – какие их особенности способствуют решению задач оптимизации; анализируются некоторые техники, которые могут улучшить обучение агента – добавление кастомной модели, ограничивающей выбор действий и разделение матрицы условий на части размера batch, и подача ее агенту. Помимо этого, приводятся основные этапы проектирования окружения для действий агента.

Как уже отмечалось выше, основными трудностями нахождения оптимальностей в задачах математического программирования являются: большая размерность данных, целочисленные ограничения. Несомненно, перед осуществлением расчетов необходимо обработать данные для представления их алгоритму обучения с подкреплением через окружение, как правило, среда представляет из себя класс с несколькими основными методами, вызываемыми агентом. Окружение необходимо агенту для получения данных из среды и обратной реакции в качестве награды. Основными составляющими среды и агента являются: пространство возможных действий, пространство наблюдений (состояний), вычисление награды, также для реализации агента с собственной моделью обучения необходимо определить бинарный вектор размерности количества действий, чтобы замораживать неэффективные действия. Также необходимо принять решение о выборе конкретного алгоритма RL среди наиболее популярных: PPO, DQN, A3C. Как известно, интеллектуальные агенты делятся на две категории: те, которые оптимизируют и обучают

политику [1], и те, которые обучают Q или V функцию полезности; кроме того, существуют алгоритмы, одновременно настраивающие и стратегию, и функцию полезности. Нашей задачей также будет показать эффективность некоторых из этих методов. Также стоит отметить, что при программировании задачи MIP необходимо определить класс – callback, который позволит отслеживать релевантные действия и запоминать значения премии на каждом шаге для выбора наилучшей.

Существует множество других подходов для решения задач дискретного программирования – это большое количество решателей, активно применяемых современными веб-сервисами, например: генетические алгоритмы, симплекс метод, метод ветвей и границ, муравьиные колонии, метод coin-or. Но относительно новой по своим характеристикам является технология обучения с подкреплением, которая не ограничивается масштабами поставленной задачи и может, при длительном обучении, решить любую проблему намного быстрее существующих версий.

Постановка задачи

При поиске экстремумов линейной функции особое место, с точки зрения сложности поиска решений, занимают задачи с требованиями целочисленности всех переменных, при этом данное условие может относиться к некоторой части переменных, тогда задача будет называться частично целочисленной. Рассмотрим каноническую запись задачи MIP.

Максимизировать целевую функцию:

$$F = \sum_{j=1}^{n} c_j x_j. \tag{1}$$

При следующих условиях и ограничениях:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, i \in \overline{1,m}, \tag{2}$$

В анализируемой работе были смоделированы 4 варианта данной задачи, отличающиеся непосредственно размерностью данных. Самый первый вариант состоит из 500 переменных и 101 ограничений; вторая задача имеет

$$x_j \geq 0, x_j \in \mathbb{N}, j \in \overline{1,n.} \tag{3}$$

размерность 1554 переменных и 311 ограничений, третья – 9756 переменных и 673 ограничений, и, наконец, четвертая – 18467 переменных и 1762 ограничений. Основную сложность данной задачи составляют большое количество переменных и их ограничения на целочисленность.

Рассмотрим методы и подходы того, как входные данные подготовить для алгоритмов RL. В процессе проведения исследований было выяснено, что для адекватной работы интеллектуальных агентов необходимо трансформировать задачу следующим образом: значения b представляются в виде вектора размера, равного числу ограничений, где каждому i-му значению соответствует строчка матрицы c, при этом на определенной позиции ij стоит значение коэффициента a_{ij}, что характеризует вышеприведенные формулы. В итоге имеются три основных компонента постановки задачи: матрица c – матрица условий, вектор b – вектор ограничений и p – вектор коэффициентов при целевой функции. Кроме того, вводится переменная *ubound*, обозначающая какие дискретные значения могут принимать переменные. Стоит отметить, что сложность могут представлять такие задачи, матрицы условий которых сильно разрежены. Конечно, для решения реальных задач дискретного программирования данные поступают, как правило, в следующих форматах: json, mips, csv. Для них необходимо обработать входные файлы и извлечь основные составляющие задач линейного программирования, а после преобразовать к матричному виду.

Проектирование универсального окружения

Одной из важных составляющих концепции машинного обучения с подкреплением является окружение или среда, где действует агент. Как правило, процесс обучения агента происходит следующим образом: имеется пространство состояний, в котором может находиться агент. Выполняя определенные действия из заданного пространства, он переходит в новое состояние и получает определенную награду, при этом выбор действия происходит в соответствии с политикой (стратегией) – обычно это функция, отображающая пространство наблюдений в пространство доступных действий, выражается распределением вероятности выбора действия [3]. Среда необходима для того, чтобы агент получал новые данные и обратные реакции в качестве награды для осуществления процесса обучения. Данная схема продемонстрирована на рисунке 1.

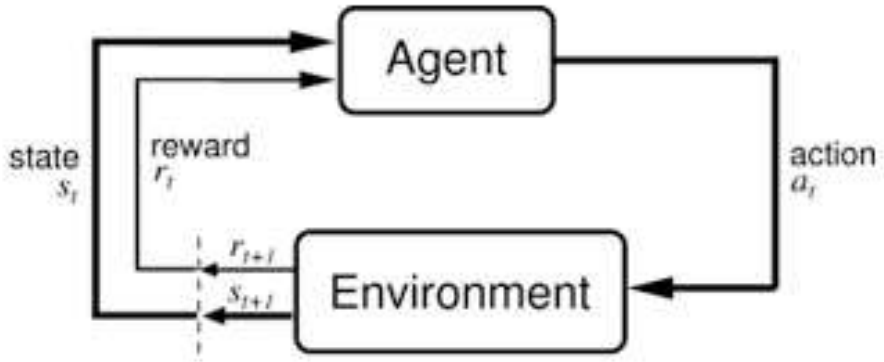


Рис.1. Схема работы агента

Существуют два основных подхода к реализации принципов обучения агента выбирать самые оптимальные действия. Во-первых – тренировать стратегию обучения, то есть использовать нейронные сети для нахождения таких параметров распределения вероятностей действий, которые приводят к максимальной ожидаемой дисконтируемой отдаче. Во-вторых, настройка функций полезности состояния и полезности состояния при определенном действии (*V, Q* функции) [2].

Для разработки окружения был использован функционал библиотеки gym, предоставляющей как и стандартные окружения, так и возможности для написания своего для конкретной задачи. Среда представляет из себя Python класс, в котором определены 3 метода: конструктор, reset – возвращает первоначальное состояние системы, step – принимает значение действия и возвращает новое состояние награду, флаг (необходим для проверки, завершился ли эпизод или нет). Дополнительно можно указать метод render, визуально демонстрирующий обучение агента.

В конструкторе класса необходимо с помощью структур данных библиотеки gym определить:

1. Пространство действий $action\ space\ = \{0,1,\ldots,ubound\}$;
2. Пространство наблюдений как хэш-таблицу с двумя ключами:

$$rem = \{b_1, b_2, \ldots, b_m\}; \quad (4)$$

$$j = \{0,1,\ldots,n\}. \quad (5)$$

Переменная j выступает в роли счетчика для перехода к новому столбцу в матрице условий (при этом происходит изменение состояния – это можно представлять как расходование доступных ресурсов, действуя жадно).

Обновление всего процесса осуществляется посредством вычитания из состояния *rem* произведения *j*-го столбца матрицы *c* на переданное действие, здесь же, если какой-либо элемент в новом векторе меньше 0, то на данном шаге присваиваем награду, равную -1, что, в свою очередь, свидетельствует о плохом выборе действия, иначе награда равна произведению полученного

методом действия на *j*-й элемент вектора коэффициентов *p*. Далее происходит сохранение новых значений.

Важным моментом при обучении своего собственного агента является добавление в окружение еще одной структуры данных – бинарного вектора длины, отражающего количество доступных действий, чтобы при умножении на логарифм вероятностей действий заморозить некоторые из выходов.

Таким образом, данный метод создания среды универсален и может быть применим к абсолютно любым задачам дискретного программирования, как с небольшими, так и без изменений параметров системы.

Обзор основных алгоритмов обучения с подкреплением

На сегодняшний день существует колоссальное количество различных алгоритмов обучения интеллектуальных агентов, каждый из которых настраивает отдельные компоненты (политика, функции ценности агентов, и т.д.), и выбор их зависит от поставленной задачи и ее особенностях.

Выше было отмечено, что все алгоритмы делятся на две большие категории: обучающие политику (PG, PPO и т.д.), настраивающие функции полезности агента (DQN, SARSA). Также отдельно от всего существуют методы, оптимизирующие как стратегию, так и функции ценности (A3C, A2C). В данной работе было обнаружено, что алгоритмы, обучающие функции ценности и смешанные, не способны находить оптимальное решение, из-за того что в рассматриваемой задаче функции *Q* и *V* в полной мере не отражают картину выбора оптимального действия, так как правило (стратегия) не меняется. Поэтому в качестве основы для построения интеллектуального агента был выбран алгоритм PPO [6], так как помимо оптимизации градиента политики он имеет много других функций, позволяющих эффективнее выстраивать стратегию обучения.

Реализация интеллектуальных агентов осуществлялась с помощью API библиотеки Ray, которая предоставляет не только набор алгоритмов RL, но и гибкие возможности для настройки программ под конкретные задачи. Прежде чем анализировать разработанные модули, стоит упомянуть о последовательности (pipe-lin) разработки:

1. Определение конфигурации модели (инициализация параметров).
2. Написание класса для сохранения состояний на каждом шаге обучения.
3. Запуск алгоритмов с выводом состояния обучения для контроля основных метрик RL (mean_episode_reward, total_episodes, mean_episode_lenght и т.д.) – подключение TensorBoard для интерактивного контроля.

Всего было реализовано три основных алгоритма, основанных на оптимизации стратегии выбора действия (PPO). Все они имеют один важный элемент, позволяющий следить за процессом обучения – задав специальную переменную, при достижении которой агент начнет запоминать награду и вычисленные действия. Таким образом, при большом числе итераций алгоритм может увеличить значение переменной, тем самым найдя больше переменных,

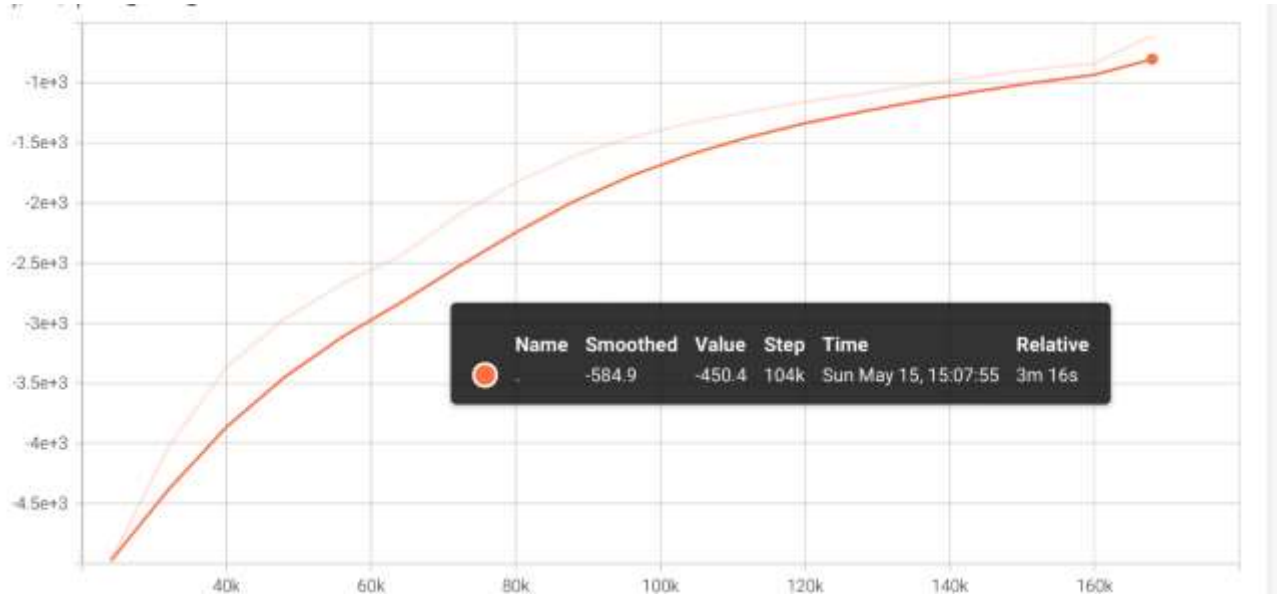
действия с которыми приводят к награде и решению, близкому к оптимальному. Кроме того, указав в параметрах конфигурации размер обучающегося набора (batch), равный числу потоков, умноженному на число переменных, можно добиться допустимого значения целевой функции и гарантии, что модель найдет необходимые действия для каждой переменной.

Как и в задачах классического машинного и глубинного обучения, можно производить поиск оптимальных гиперпараметров с помощью API Ray Tune. Данный подход может быть разумным к применению, когда имеется достаточно много времени на решение сложной задачи оптимизации, так как требует, по понятным причинам, временных затрат, но при этом могут быть найдены лучшие варианты. Рассмотрим особенности интеллектуальных агентов:

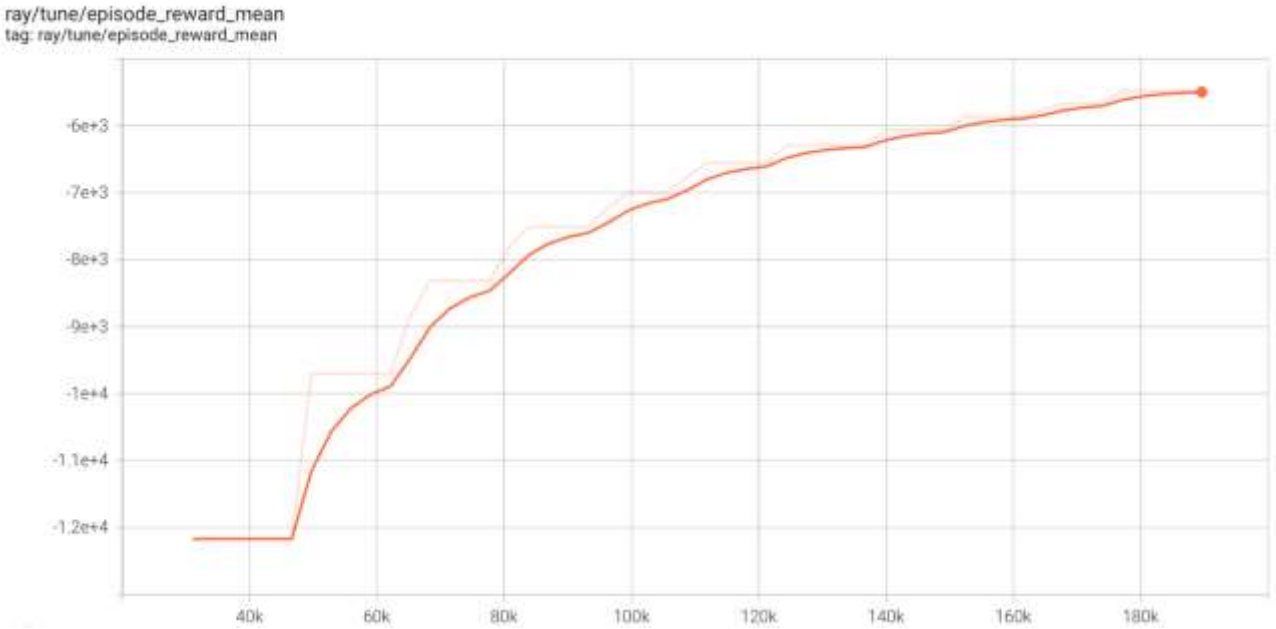
- Обычный PPO с классом для сохранения – это самый распространённый вариант среди разработанных. У него хорошее время работы и адекватные решения многомерных оптимизационных задач.
- PPO с дополнительной моделью для замораживания неоптимальных действий. Как показали результаты, этот вариант работает быстрее предыдущего благодаря тому, что в методе forward агента происходит перемножение логарифмов вероятностей действия на бинарную маску, и итоговый массив логарифмов вероятностей содержит много нулевых значений, которые позволяют запретить нерациональные действия агенту.
- Техника обучения по mini-batch. Основная ее суть заключается в том, что перед подачей данных алгоритму необходимо создать массив индексов с шагом, равным mini-batch. В основном цикле обучения происходит следующее – на каждом шаге берем срез матрицы условий, вектора границ, вектора коэффициентов, обучаем агента на данной выборке и в конце сохраняем веса модели, а также обновляем вектор границ путем перемножения вектора полученных действий на матрицу условий, тем самым сокращая пространство возможных состояний (истощение ресурсов), кроме того, на каждой итерации необходимо устанавливать веса, рассчитанные ранее, чтобы тем самым обучать только одного агента и накапливать полученный опыт.

Данный подход может быть актуален и полезен, когда задача настолько большая, что не может поместиться в память компьютера. Помимо этого, в данном подходе можно не настраивать параметры для параллельного вычисления, а в качестве специальной переменной для логирования взять размер mini-batch. Возможно использование настраиваемого (custom) агента для ускорения процесса обучения.

Для более детального понимания происходящего рассмотрим графики основных метрик обучения трех моделей на соответствующих наборах данных и проанализируем их.



Puc.2. MeanEpisodeReward, SimpleAgent



Puc. 3. MeanEpisodeReward, CustomAgent

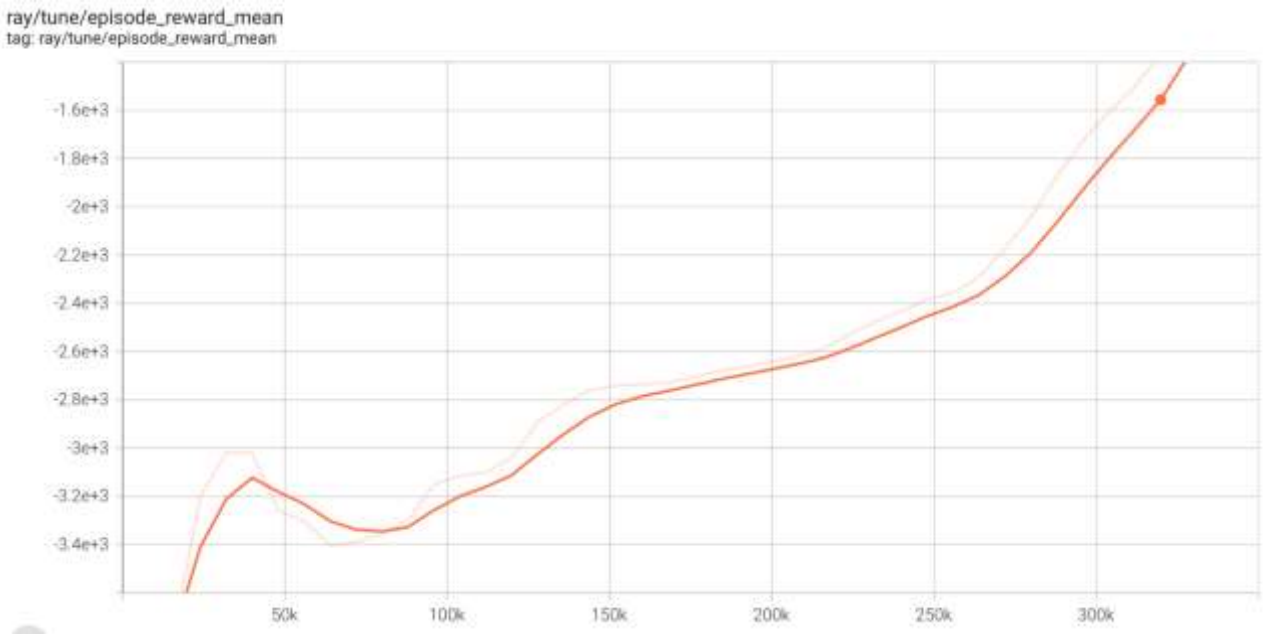


Рис. 4. MeanEpisodeReward, BatchAgent

На рисунке 2 показана метрика (mean episode reward) работы обычного PPO алгоритма, рисунок 3 демонстрирует результаты PPO с адаптированной моделью, и, наконец, рисунок 4 – результаты агента с mini-batch. Данные для задач, соответственно: Simple Agent - 1554 переменных и 311 условий, Custom Agent - 9756 переменных и 673 ограничения и Batch Agent - 12653 переменных и 673. Результаты фиксировались с помощью интерактивного инструмента – TensorBoard, и для алгоритма Batch Agent было создано столько графиков, каков зафиксировался размер batch – в целом по времени они все работают одинаково.

Анализируя полученные результаты, можно сделать вывод, что быстрее всего обучается и находит оптимальное решение Custom Agent, надежней всего оказался Simple Agent, а при поиске решений задачи в колоссальных размерностях подойдет Batch Agent.

Результаты работы моделей и сравнение их между собой

Эксперимент проводился следующим образом: каждый из рассмотренных алгоритмов решал соответствующую задачу (Размерность 1 – 500 переменных и 101 ограничение, Размерность 2 – 1554 переменных и 311 ограничений, Размерность 3 – 9756 переменных и 673 ограничения, Размерность 4 – 18467 переменных и 1762 ограничения). Основными показателями эффективности работы являлись: значение целевой функции, время работы агентов. Кроме того, производилось сравнение разработанных моделей с алгоритмом ветвей и границ, реализованным в библиотеке OR-Tools.

Таблица 1

Результаты работы

Модель	Размерность 1	Размерность 2	Размерность 3	Размерность 4
Simple Agent	Objective value 943.5 Время работы 3 минуты	Objective value 1368 Время работы 5 минут	Objective value 7532.9 Время работы 8 минут	Objective value 16536 Время работы 11.30 минут
Custom Agent	Objective value 901 Время работы 1.45 минуты	Objective value 1400 Время работы 3 минуты	Objective value 7456.6 Время работы 5 минут	Objective value 15532.65 Время работы 8 минут
Batch Agent	-	-	-	Objective value 16435 Время работы 20 минут
OR-Tools	Objective value 950 Время работы 33 минуты	За 60 минут не нашел решение	За 60 минут не нашел решение	За 60 минут не нашел решение

Исходя из данных таблицы, заметим, что последняя модель не может справиться с нахождением даже допустимых решений задачи смешанно-целочисленного линейного программирования. Модель Batch Agent оценивалась только по задаче с большой размерностью, так как данный вид алгоритмов эффективнее применять тогда, когда данные не помещаются в память, кроме того, было выбрано достаточно большое значение количества необходимых итераций: этот параметр можно также регулировать, чтобы настроить скорость работы интеллектуального агента. Самым быстрым оказался Custom Agent, из-за добавления возможности замораживать неэффективные действия. Данный алгоритм может быть использован в системах поддержки принятия решения в качестве решателя задач смешанного целочисленного линейного программирования. Стоит также отметить, что основную сложность для поиска решений задачи могут составлять строчки матрицы условий, содержащие разреженные бинарные значения.

В качестве демонстрации работы данной модели рассмотрим ее результаты работы на рисунках 5 и 6 со случайно сгенерированными данными большой размерности.

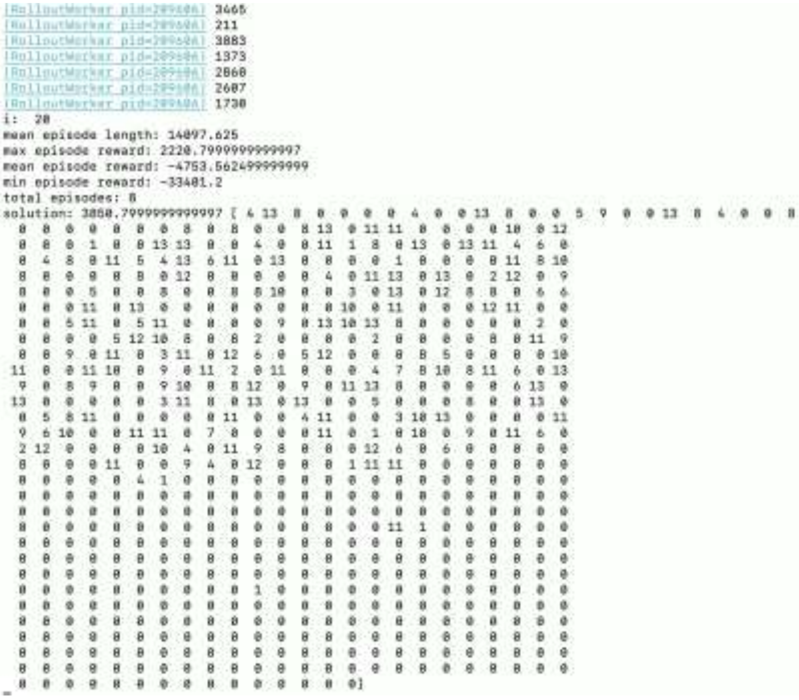


Рис. 5. Результат № 1 работы агента

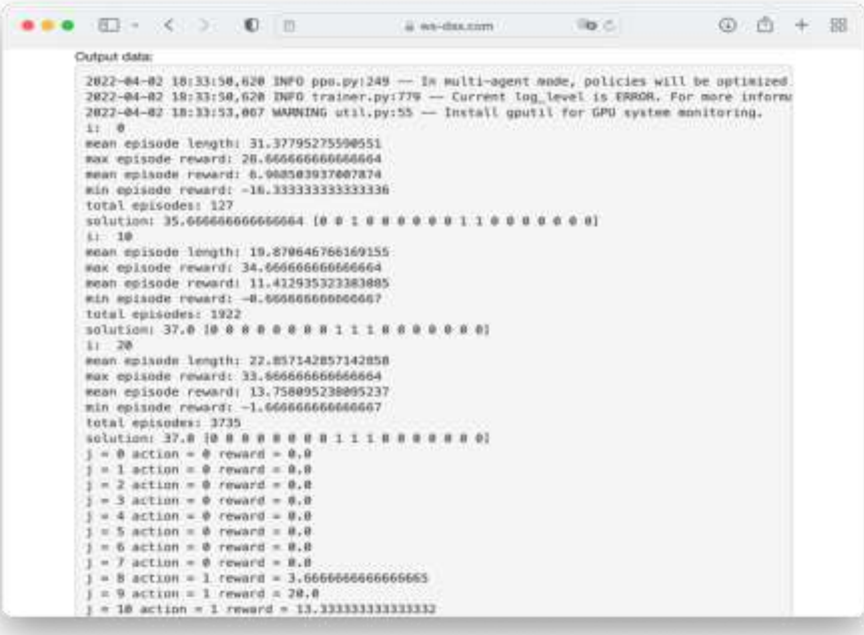


Рис. 6. Результат № 2 работы алгоритма на тестовой выборке

Заключение

Были разработаны модели интеллектуальных агентов на основе методов машинного обучения с подкреплением для решения задач смешанного целочисленного программирования больших размерностей. Следует отметить несколько важных моментов общей концепции решения задач подобного рода:

- Во-первых, необходимо правильно подготовить данные для подачи их на вход модели. Так как в обучении с подкреплением основными элементами являются окружение и алгоритм работы, то данные следует представить в нотациях: состояние, действие, отдача, следующее состояние.
- Во-вторых, следует выбирать алгоритм обучения агента, исходя из опытных или теоретических соображений. Эффективнее производить оценку по функции ценности или по политике модели; именно тут и используются нейросетевые методы в качестве построения распределения вероятностей. Кроме того, можно применять другие эвристики для решения задач: сохранять значения на каждом шаге, пробовать собственные функционалы и накладывать штрафы в виде ограничений на те или иные значения.

Данная разработка может использоваться в качестве нового решателя в интеллектуальных системах поддержки принятия решений для задач высокой размерности.

Библиографический список

1. Алфимцев А.Н. Мультиагентное обучение с подкреплением. Учебное пособие. — М.: МГТУ им Н. Э. Баумана, 2021. — 220 с.: ил.

2. Лаура Г., Лун К.В. Глубокое обучение с подкреплением: теория и практика на языке Python. – СПб.: Питер, 2022. – 416 с.: ил. (Серия “Библиотека программиста”).

3. Судаков В.А., Сивакова Т.В. Мультиагентное моделирование на базе нечетких суждений экспертов области борьбы с пандемией. // В книге: Математическое моделирование. Тезисы II Международной конференции. Москва. — 2021. — С. 81-83.

4. Богданов И.П., Судаков В.А., Топоров Н.Б. Оптимизация загрузки упорядоченной совокупности летательных аппаратов // Мат. моделирование, № 4, 2020. С.43 – 56.

5. Hui J. RL – Proximal Policy Optimization (PPO) Explained. https://jonathan-hui.medium.com/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12

Оглавление