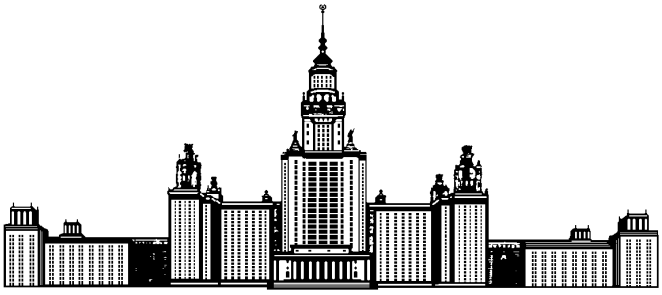

Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

Фролов Антон Алексеевич

Комбинаторная оптимизация с помощью нейронных сетей

Combinatorial Optimization using Neural Networks

КУРСОВАЯ РАБОТА

Научный руководитель:
Д. А. Кропотов

Москва, 2022

Содержание

1 Введение **2**

2 Постановка задачи **2**

3 Нейросетевой подход **2**

3.1 Этапы построения модели 2

3.2 Структура модели 3

3.3 Обучение ADNN 4

3.4 Эксперименты 5

4 Метод ветвей и границ **7**

4.1 Стратегии ветвления 8

4.2 Аппроксимация целочисленного решения 9

4.3 Пример 9

4.4 Эксперименты 10

5 Заключение **11**

1 Введение

Задачи комбинаторной оптимизации зачастую являются NP-трудными и для них не существует эффективных алгоритмов решения. В общем случае для нахождения оптимального решения необходимо перебрать все возможные значения переменных, количество комбинаций которых экспоненциально зависит от размерности задачи. Поэтому для таких задач применяются различные эвристики, помогающие в среднем уменьшить время поиска оптимального решения, однако не гарантирующие быструю сходимость на любых входных данных. Примером такой эвристики может послужить метод ветвей и границ, который мы рассмотрим далее.

Эффективность нейронных сетей во многих областях машинного обучения делает их привлекательными для решения задач комбинаторной оптимизации и в частности задач целочисленного программирования. Нейронные сети могут решать поставленную задачу как целиком, получая на вход условия задачи, и выдавая ответ на выходе [3, 4], так и встраиваться в существующие алгоритмы оптимизации, ускоряя их [2]. В частности, была предложена эффективная архитектура, решающая задачу коммивояжера [3].

В этой работе будет рассмотрена одна из задач целочисленного программирования, подмножества задач комбинаторной оптимизации.

2 Постановка задачи

Рассмотрим следующую задачу целочисленного программирования:

$$\underset{x}{\text{minimize}}\ f(Hx - y)$$

$$\text{subject to } x_n \in \mathcal{A} \subset \mathbb{Z}$$

где $H \in \mathbb{R}^{Q \times N}$ — матрица некоторого преобразования, $y \in \mathbb{R}^Q$ — вектор наблюдаемых значений, $f(\cdot)$ — функция потерь, $x \in \mathbb{R}^N$ — искомый вектор, $\mathcal{A}$ — множество целочисленных значений, которые могут принимать элементы x_n вектора x.

3 Нейросетевой подход

Авторы статьи Learning for Integer-Constrained Optimization through Neural Networks with Limited Training [4] предлагают следующий нейросетевой подход для решения поставленной задачи.

3.1 Этапы построения модели

Построение модели состоит из следующих этапов:

1. Определяется вероятностная модель, в которой отклонение $r = Hx - y$ генерируется из распределения $P(r)$, определяемому по функции потерь $f(\cdot)$.

   Например, для квадратичной функции потерь $f(r) = r^2 = (Hx - y)^2$ ошибка r будет иметь нормлаьное распределение $P(r) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{r^2}{2\sigma^2}\right)$.

2. Генерируется обучающая выборка из K пар (x, r) из распределений $\mathcal{U}(\mathcal{A}^N)$ и $P(r)$ соответственно:
$$\left\{\left(x^{(1)}, r^{(1)}\right), \left(x^{(2)}, r^{(2)}\right), \ldots, \left(x^{(K)}, r^{(K)}\right)\right\}$$

3. Для фиксированной матрицы H по паре векторов (x, r) однозначно определяется вектор $y = Hx - r$. Таким образом, обучающую выборку можно представить в виде набора пар (x, y):
$$\left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(K)}, y^{(K)}\right)\right\}$$

4. Затем мы обучаем модель $\mathcal{N}$, причем $y^{(k)}$ подается на вход, а $x^{(k)}$ является желаемым выходом.

5. Обучив модель, мы можем подать ей на вход вектор y исходной задачи, получив на выходе предсказанное решение $\hat{x}$.

3.2 Структура модели

Модель $\mathcal{N}$ состоит из N различных нейронных сетей одинаковой архитектуры, каждая из которых независимо приближает n-ую компоненту вектора x.

Основываясь на введенной вероятностной модели мы можем записать плотность условного распределения $p(x|y)$ в следующем виде:

$$p(x_1, x_2, , \ldots, x_N|y) = p(x_1|y) \cdot p(x_2|x_1 y) \cdot \ldots \cdot p(x_N|x_1, \ldots, x_{N-1}, y)$$

Предположим, что элементы вектора x не зависят друг от друга:

$$p(x_1, x_2, , \ldots, x_N|y) \approx p(x_1|y) \cdot p(x_2|y) \cdot \ldots \cdot p(x_N|y)$$

В таком случае принцип максимума правдоподобия принимает следующий вид:

$$\begin{gathered}\max_{x \in \mathcal{A}^N} p(x_1, x_2, , \ldots, x_N|y) \approx \\ \approx \max_{x \in \mathcal{A}^N} \big(p(x_1|y) \cdot \ldots \cdot p(x_N|y)\big) = \\ = \prod_{n=1}^{N} \max_{x_n \in \mathcal{A}} p(x_n|y)\end{gathered}$$

Для удобства изложения возьмем следующее множество $\mathcal{A}$:

$$\mathcal{A} = \big\{-2M-1, -2M+1, \ldots, -1, 1, \ldots, 2M-1, 2M+1\big\}$$

Для нахождения $\arg\max_{x_n \in \mathcal{A}} p(x_n|y)$, $n = \overline{1, N}$ будем использовать N нейросетей $\mathcal{N}_n$, называемых ADNN (Atomic Decision Neural Network), каждая из которых будет предсказывать отношения вероятностей гипотез $\{H_0^{(2m)} : x_n = -2m+1\}$ и $\{H_1^{(2m)} : x_n = -2m-1\}$, где $m = -M, \ldots, M$:

$$L_{+-}^{(2m)}(y) = \frac{p(x_n = -2m+1|y)}{p(x_n = -2m-1|y)}$$

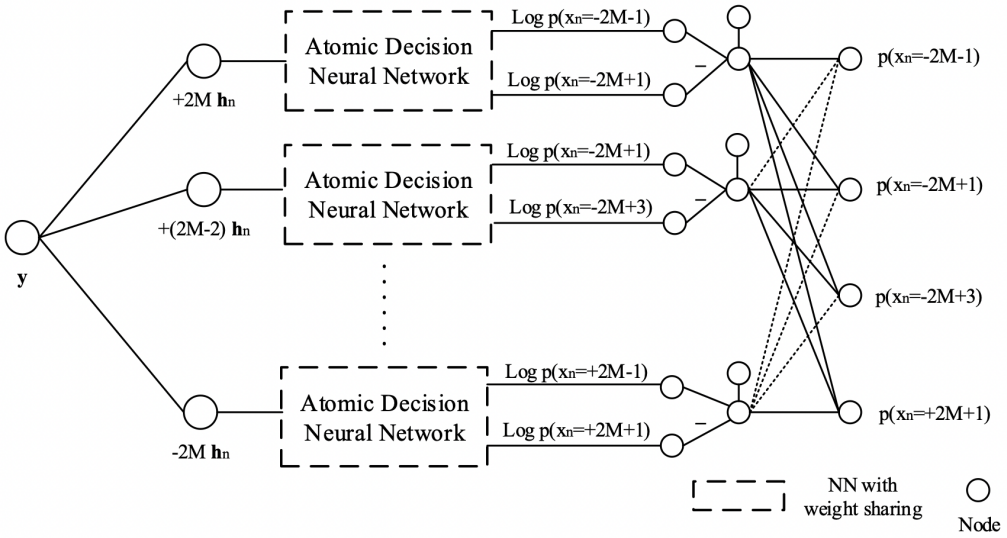

Рис. 1: Структура нейронной сети

На вход сети $\mathcal{N}_n$ будет подаваться $2M+1$ сдвинутых на $2m,\ m=-M,\ldots,M$ векторов y (Рис 1). Такой сдвиг обусловлен тем, что:

$$L_{+-}^{(2m)}(y)=\frac{p(x_n=-2m+1|y)}{p(x_n=-2m-1|y)}=\frac{p(x_n=1|y+2mh_n)}{p(x_n=-1|y+2mh_n)}=L_{+-}^{(0)}(y+2mh_n)$$

После того как получены все $L_{+-}^{(2m)}(y),\ m=\overline{-M,M}$, вероятность $p(x_n=-2m+1|y)$ можно получить по следующей формуле:

$$p(x_n=-2m+1|y)=p(x_n=-2M-1|y)\prod_{m'=M}^{m}L_{+-}^{(-2m')}(y)$$

(Предполагается, что $p(x_n=-2M-1|y)=const$)

Каждая из ADNN $\mathcal{N}_n$ представляет собой многослойный перцептрон, выходом которого являются два числа $\log(p(x_n=1|y))$ и $\log(p(x_n=-1|y))$. В исследовании использовался перцептрон с одним скрытым слоем.

3.3 Обучение ADNN

Учитывая структуру нашей модели можно составить следующий набор данных для каждой из ADNN $\mathcal{N}_i$:

$$\left\{\left(+1,\tilde{y}_+^{(1)}\right),\left(-1,\tilde{y}_-^{(1)}\right),\ldots,\left(+1,\tilde{y}_+^{(K)}\right),\left(-1,\tilde{y}_-^{(K)}\right)\right\}$$

где

$$\tilde{y}_+^{(k)}=y^{(k)}-x_n^{(k)}h_n+h_n$$
$$\tilde{y}_-^{(k)}=y^{(k)}-x_n^{(k)}h_n-h_n$$

В качестве функции потерь для обучения ADNN выбирается бинарная кросс-энтропия:

$$\text{LogLoss}(x)=\mathbb{I}\big[x=1\big]\log\big(p(x=1|y)\big)+\mathbb{I}\big[x=-1\big]\log\big(p(x=-1|y)\big)$$

3.4 Эксперименты

В качестве метрики качества в последующих экспериментах будем использовать нормализованный на величину вектора y MSE:

$$L(x) = \frac{\|Ax - y\|^2}{\|y\|^2}$$

Посмотрим, как ведет себя модель в зависимости от размерности задачи. (Рис. 2)

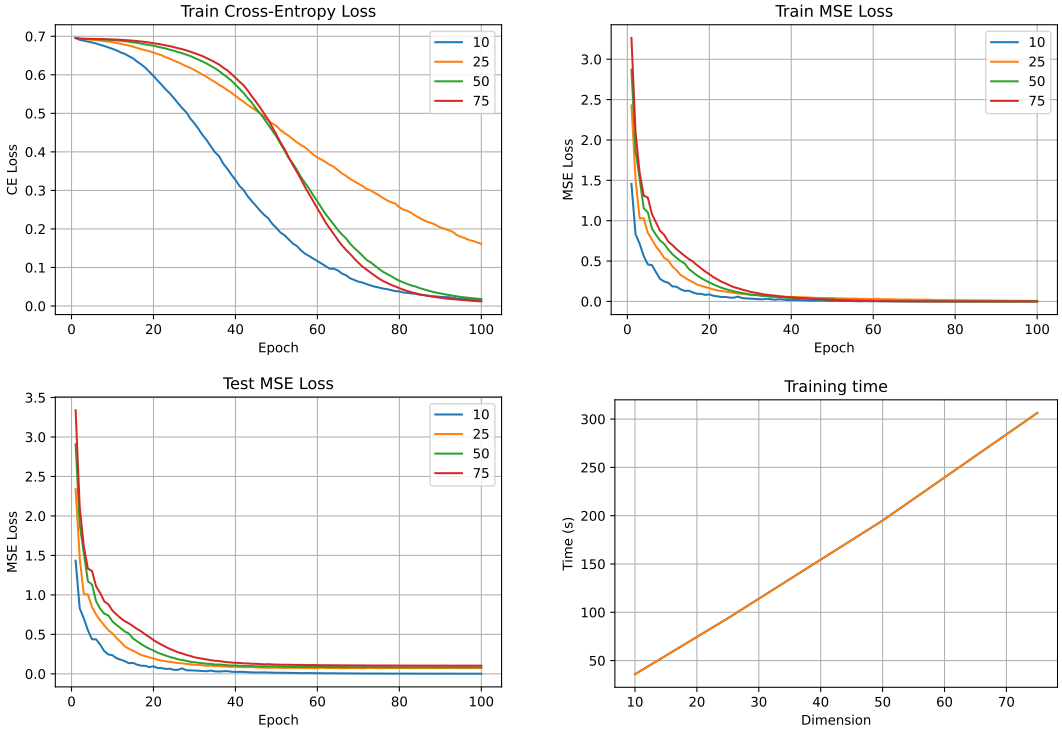

Рис. 2: Графики зависимостей от размерности задачи

Можно заметить, что для всех размерностей MSE сходится к 0 раньше, чем сходится кросс-энтропия. С ростом размерности задачи MSE сходится медленнее. Время обучения линейно зависит от размерности задачи, так как для задачи размерности N необходимо обучить N нейросетей ADNN.

Посмотрим, как ведет себя модель в зависимости от размера обучающей выборки. (Рис. 3)

Можно заметить, что MSE для всех выборок сходится к 0, однако для выборок размером меньше 128 ошибка на тестовой выборке все еще значительная, данных для решения задачи не достаточно. Можно сказать, что преимуществом модели является небольшой размер выборки, необходимой для обучения.

Далее посмотрим, как ведет себя модель в зависимости от размера множества $\mathcal{A}$. (Рис. 3)

Из графиков видно, что модель хорошо справляется с небольшими задачами, но при увеличении размера множества начинает работать хуже.

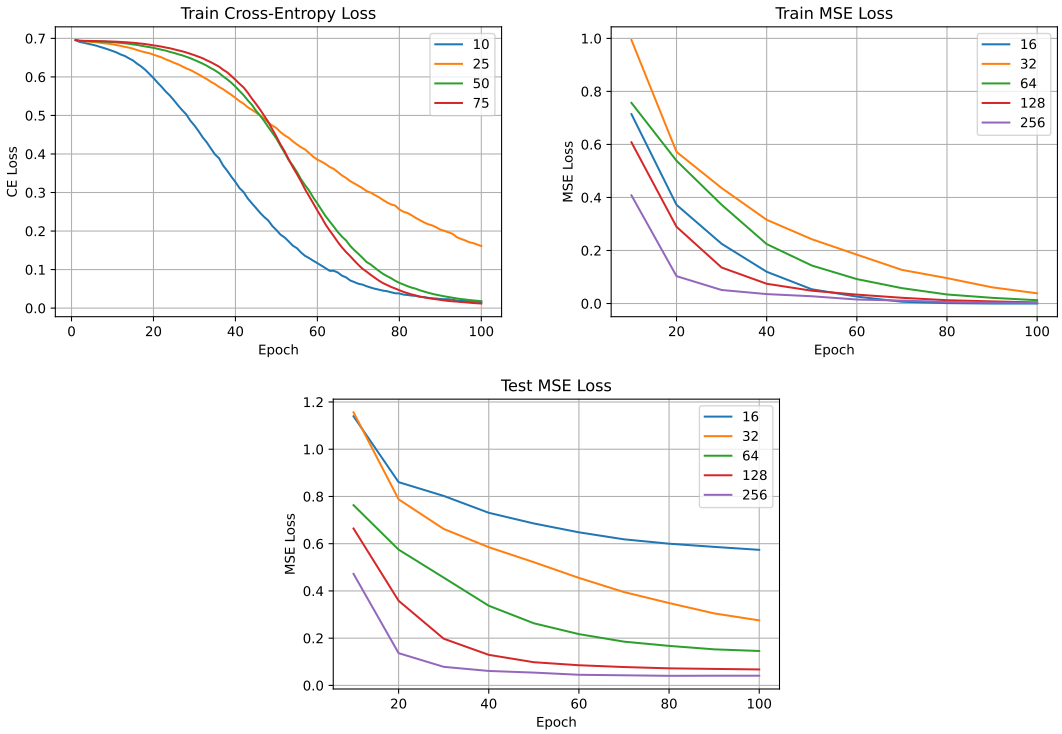

Рис. 3: Графики зависимостей от размера обучающей выборки

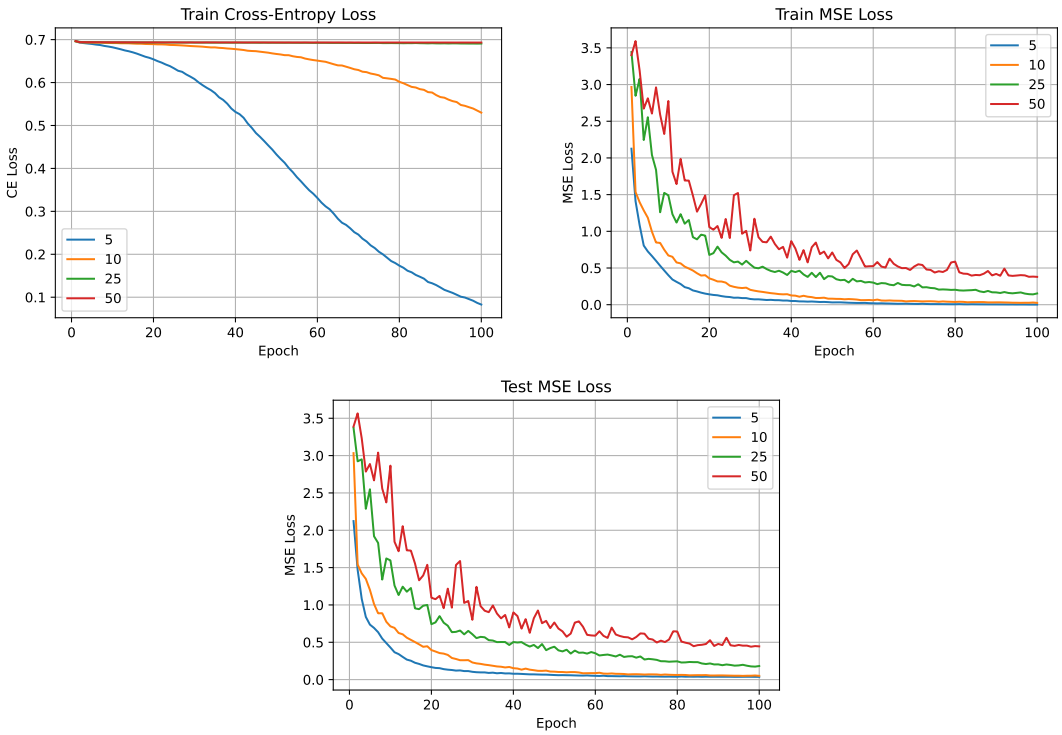

Рис. 4: Графики зависимостей от размера множества $\mathcal{A}$

4 Метод ветвей и границ

У описанной нейросетевой архитектуры есть недостаток. Такая архитектура не позволяет позволяет эффективно решать задачу для различных матриц H. Для каждой новой матрицы сети придется переобучать, что затратно по времени.

В попытке решить эту проблему был использован известный метод ветвей и границ (**branch and bound**). Этот метод подходит для решения целого спектра зада целочисленного программирования и комбинаторной оптимизации.

Метод ветвей и границ основан на предположении, что множество всех возможных решений может быть разбито на несколько меньших подмножеств. Затем эти подмножества оцениваются на оптимальность, и процедура разбиения повторяется рекурсивно до тех пор, пока не будет найдено оптимальное решение.

В процессе выполнения алгоритма строится дерево решений. Корнем в этом дереве является исходная задача:

$$\text{minimize } f(Ax - y)$$
$$\text{subject to } x \in \mathcal{A} \subseteq \mathbb{Z}$$

Затем для этой задачи отбрасываются ограничения на целочисленность переменных и находится решение непрерывной задачи:

$$\text{minimize } f(Ax - y)$$
$$\text{subject to } x \in \mathbb{R}^n$$

Обозначим решение непрерывной задачи как x^*. Значение функционала $f(Ax^* - y)$ является его нижней границей. Далее мы будем отслеживать нижние границы в текущем наборе узлов, выбирая из них тот, что имеет наименьшую нижнюю границу (наиболее оптимален).

Затем мы выбираем переменную с индексом i, по которой будем ветвить текущую задачу и разбиваем текущую задачу на две. В первой задаче добавим к текущим ограничениям ограничение $x_n \leq \lfloor x_n^* \rfloor$, во второй — ограничение $x_n \geq \lceil x_n^* \rceil$. Далее решим непрерывные варианты полученных задач и получим две новые нижние оценки в новых вершинах.

Следующей вершиной для ветвления выберем ту, на которой достигается минимум нижней оценки.

Если для вершины при поиске непрерывного решения оказалось, что непрерывное решение совпадает с целочисленным, мы обновляем глобальную верхнюю оценку нашей задачи. Если верхняя оценка совпадает с нижней оценкой, то найдено оптимальное целочисленное решение. Иначе мы отбрасываем все те узлы, нижняя оценка которых выше найденной верхней оценки.

Далее мы повторяем процедуру ветвления для узла с наименьшей нижней оценкой до тех пор, пока не найдем оптимальное целочисленное решение.

Итак, метод ветвей и границ можно записать в виде следующей последовательности шагов:

0. Решить непрерывный аналог исходной задачи в корневой вершине и получить нижнюю границу минимизируемого функционала. Инициализировать список текущих вершин корневой вершиной.

1. Выбрать из списка текущих вершин вершину с наименьшей нижней границей.

2. Выбрать переменную x_i для ветвления и осуществить ветвление.

3. Получить решение непрерывных задач в двух новых вершинах, посчитать нижние границы и добавить вершины в список текущих вершин.

4. Если непрерывное решение в одной из вершин совпадает с целочисленным:

   (a) Положить верхнюю границу равной минимуму из текущей и найденной верхней границы.

   (b) Удалить из списка текущих вершин все вершины, нижняя граница для которых больше обновленной верхней границы.

5. Перейти к пункту 1.

4.1 Стратегии ветвления

На этапе 2 необходимо выбрать переменную, по которой будет осуществляться ветвление. Правильно выбирая переменную для ветвления можно значительно уменьшить размер дерева поиска решения.

В качестве тривиальной эвристики можно предложить округлять непрерывное решение и выбирать переменную с наибольшей дробной частью (наибольшим модулем отклонения между значением в непрерывном и округленном решении). Можно сказать, что преимуществом такой эвристики является ее скорость, нужно всего лишь округлить непрерывное решение.

Однако, такая эвристика не очень эффективна и существует ряд других, более продвинутых методов ветвления. В качестве примера такого метода мы рассмотрим метод сильного ветвления (**strong branching**) [1]. В этом методе для выбора переменной сначала осуществляются все возможные ветвления по переменным x_i, в каждом из них находятся новые нижние оценки функционала z_i^- для округления вниз и z_i^+ для округления вверх. Качество полученного ветвления оценивается следующей величиной:

$$SB_i = \max\left\{z_i^+ - z_i, \varepsilon\right\} \cdot \max\left\{z_i^- - z_i, \varepsilon\right\}$$

где $\varepsilon = 10^{-6}$.

Такая стратегия ветвления требует решения $O(N)$ непрерывных задач при каждом ветвлении, однако потенциально может уменьшить размер дерева поиска решения.

4.2 Аппроксимация целочисленного решения

Этап 4 можно выполнять не только, когда целочисленное решение полностью совпадает с непрерывным. В каждом узле можно приметить алгоритм, который по текущему состоянию задачи и ее непрерывному решению будет предлагать некоторое целочисленное решение. По этому решению можно так же обновить верхнюю границу и отбросить вершины, у которых нижняя граница выше новой верхней границы.

В качестве тривиальной стратегии можно выбрать округление непрерывного решения до целочисленного. Используются и более сложные стратегии, однако здесь они рассмотрены не будут.

4.3 Пример

Приведем пример решения небольшой задачи с помощью метода ветвей и границ.
Возьмем следующие матрицу A и вектор y:

$$A = \begin{bmatrix} 3 & 5 \\ -2 & 7 \end{bmatrix} \quad y = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$$

В качестве функционала $f(\cdot)$ возьмем квадрат нормы разности $\|Ax - y\|^2$.
Дерево поиска решения изображено на рисунке 5.

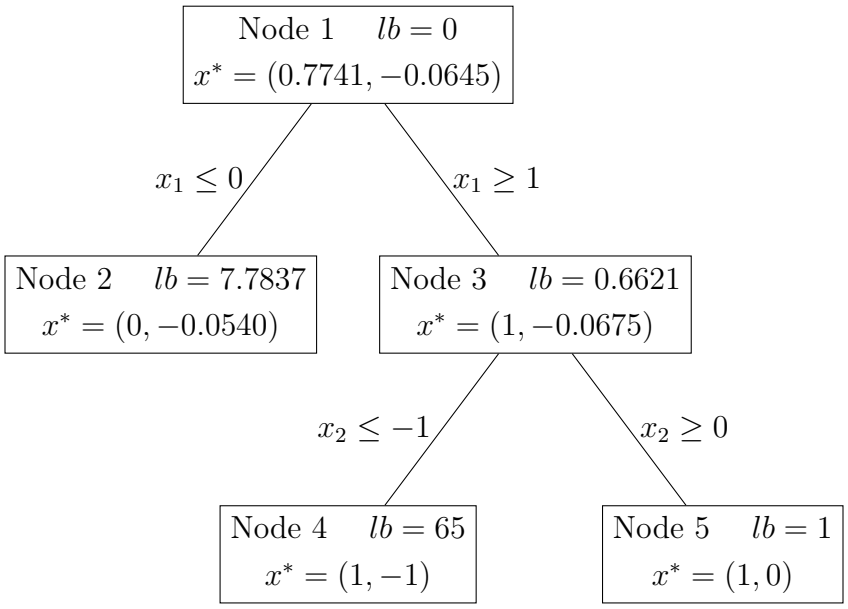

Рис. 5: Пример дерева поиска решения

После решения исходной непрерывной задачи в вершине Node 1 происходит ветвление по переменной x_1, и появляются два новых узла Node 2 и Node 3, которые добавляются в список текущих узлов. Нижняя граница меньше у Node 3, поэтому следующее ветвление осуществляется в нем (по переменной x_2), и в список текущих узлов добавляются два новых узла Node 4 и Node 5. Наименьшая нижняя граница из текущих вершин у вершины Node 5, кроме того в ней непрерывное решение совпадает с целочисленным, а значит найдено оптимальное целочисленное решение.

4.4 Эксперименты

Очевидным минусом метода ветвей и границ является то, что размер дерева будет быстро возрастать с увеличением размерности задачи. Как уже было сказано, эту проблему можно попытаться решить с помощью эффективной стратегии ветвления.

Посмотрим, как зависит количество вершин дерева поиска решение от размерности задачи. Приведем графики как для стратегии максимума дробной части, так и для метода strong branching при фиксированном размере множества $\mathcal{A} = \{-M, \dots, M\}$, $M = 10$. (Рис. 6)

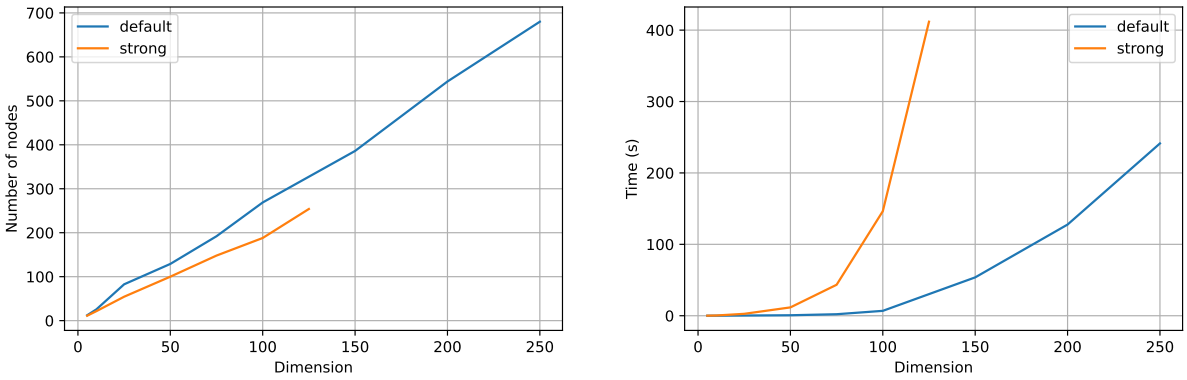

Рис. 6: Графики зависимости времени и количества вершин от размерности задачи

Можно заметить, что количество узлов зависит линейно от размерности задачи. В то же время зависимости времени от размерности больше, чем линейная. Это связано с тем, что в каждой вершине нужно решать непрерывную задачу, решение которой также зависит от размерности задачи.

Также можно заметить, что, как и ожидалось, стратегия strong branching уменьшает число узлов в дереве. Однако перебор всех возможных ветвлений занимает много времени.

Далее исследуем зависимость количества вершин и времени от размера целочисленного множества $\mathcal{A}$ при фиксированной размерности $N = 20$. (Рис. 7)

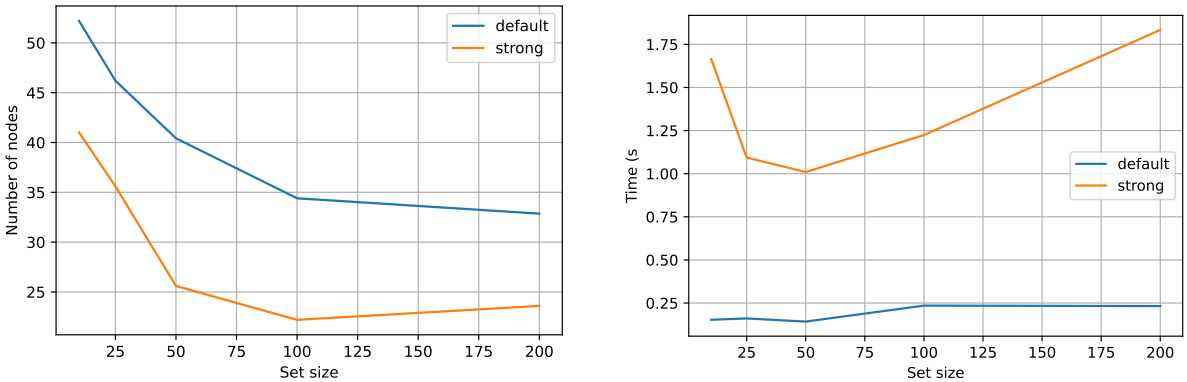

Рис. 7: Графики зависимости времени и количества вершин от размера множества $\mathcal{A}$

Можно сказать, что размер множества $\mathcal{A}$ не оказывает значительного влияния ни на время работы, ни на количество вершин в дереве поиска решения. Количество вершин

даже немного уменьшается с ростом $|\mathcal{A}|$, возможно задача становится проще.

5 Заключение

Таким образом, в этой работе было предложено два метода решения поставленной задачи. Первый метод использует нейронные сети, и эффективно работает на задачах небольшой размерности с фиксированной матрицей H. Для решения задач с различными матрицами был предложен метод ветвей и границ, классический метод комбинаторной оптимизации, гарантированно находящий оптимальное решение, однако проигрывающий по времени работы.

В перспективе возможно ускорение метода ветвей и границ с помощью машинного обучения. Например для ускорения метода ветвей и границ, примененного к задаче линейного программирования, авторами статьи [2] было предложено использовать графовые сверточные сети. В дальнейшем было бы интересно попробовать перенести предложенный ими подход на рассмотренную нами задачу.

Исходный код моделей, описанных в работе, можно найти по ссылке.

Список литературы

[1] Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 724–731. AAAI Press, 2016.

[2] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks, 2020.

[3] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2015.

[4] Zhou Zhou, Shashank Jere, Lizhong Zheng, and Lingjia Liu. Learning for integer-constrained optimization through neural networks with limited training, 2020.