

1 Аннотация

Целью данной работы является изучение конструкции Т-многообразий сложности 1 в терминах CDP (комбинаторных дивизориальных многогранников) и компьютерная реализация этих многообразий и некоторых их свойств. В данной работе был разработан и реализован алгоритм проверки CDP на эквивалентность, а также была реализована генерация CDP из многогранников.

Содержание

1	Аннотация	1
2	Введение	3
2.1	CDP и T-многообразия	3
2.2	Преобразования CDP	4
2.3	Эквивалентность	4
2.4	Генерация CDP из многогранников	5
3	Компьютерная реализация CDP	6
3.1	Преобразования CDP	9
3.2	Алгоритм проверки эквивалентности	11
3.2.1	Описание	11
3.2.2	Примеры работы:	13
3.2.3	Доказательство корректности	15
3.3	Алгоритм генерации CDP из многогранников	18
3.3.1	Примеры работы алгоритма	18
4	Приложение	21
4.1	piecewise_affine_function.py	21
4.2	cdp.py	24
4.3	generate_cdp.py	30

2 Введение

2.1 CDP и Т-многообразия

Пусть задана решетка M , \square - многогранник с вершинами в узлах решетки, \square° - внутренность многогранника, $\partial\square$ - его граница.

Определение 2.1 (Дивизориальный многогранник). Дивизориальный многогранник на гладкой проективной кривой Y относительно решетки M - это кусочно-линейная вогнутая функция $\Psi = \sum_{P \in Y} \psi_P \dot{P} : \square \rightarrow \text{Div}_{\mathbb{Q}} Y$, область определения которой - многогранник \square на решетке M , область значений - подмножество группы \mathbb{Q} -дивизоров на Y , такая что

1. $\deg \Psi(u) > 0$ для $u \in \square^\circ$
2. $\deg \Psi(u) > 0$ или $\Psi(u) \sim 0$ для $u \in \square$
3. График ψ_P имеет целочисленные вершины для всех $P \in Y$

, где $\deg \Psi(u) = \sum_{P \in Y} \psi_P$

Определение 2.2 (Т-многообразие). Т-многообразием сложности k называется нормальное многообразие X над \mathbb{C}_n эффективным действием тора $T \simeq (\mathbb{C}^*)^{n-k}$

Дивизориальные многогранники соответствуют рациональным поляризованным Т-многообразиям сложности 1 ([3]).

Определение 2.3 (CDP). Комбинаторный дивизориальный многогранник (combinatorial divisorial polytope) относительно решетки M состоит из многогранника $\square \subset M \otimes \mathbb{R}$ и кортежа (ψ_1, \dots, ψ_n) кусочно-линейных вогнутых функций $\psi_i : \square \rightarrow \mathbb{R}$ таких что

- 1 Для всех i граф ψ_i - полиэдральный комплекс с целочисленными вершинами
- 2 $\forall u \in \square^\circ \sum_{i=1}^n \psi_i(u) > 0$

\square называется базой CDP

Если прикрепить каждую из функций ψ_i к точке P_i на кривой \mathbb{P}^1 (проективная прямая) - получится дивизориальный многогранник на \mathbb{P}^1 ([2]), что соответствует рациональным поляризованным Т-многообразиям сложности 1.

2.2 Преобразования CDP

Пусть дан исходный CDP с набором функций (ψ_1, \dots, ψ_n) и базой \square

Определение 2.4 (Трансформация базы (transformation of the base)). Пусть ϕ - линейное обратимое преобразование решетки M , тогда CDP с базой $\phi(\square)$ и набором функций $\psi_i \circ \phi^{-1}$ получен из исходного с помощью трансформации базы.

Определение 2.5 (Скашивание (shearing)). Пусть $v \in M^*, \beta_1, \dots, \beta_n \in \mathbb{Z}$, тогда CDP с базой \square и набором функций $u \mapsto \psi_i(u) + \beta_i \langle u, v \rangle, \forall u \in \square$, $\sum_{i=1}^n \beta_i = 0$, получен из исходного с помощью скашивания.

Определение 2.6 (Перенос (translation)). Пусть $\alpha_1, \dots, \alpha_n \in \mathbb{Z}, \sum_{i=1}^n \alpha_i = 0$, тогда CDP с базой \square и набором функций $\psi_i + \alpha_i$ получен из исходного с помощью переноса.

2.3 Эквивалентность

Определение 2.7 (Эквивалентные CDP). CDP, полученный из исходного с помощью преобразований скашивания, переноса и трансформации базы, примененных в любой последовательности и в любых количествах, эквивалентен исходному.

Определение 2.8 (Эквивариантный изоморфизм). Пусть дана группа, X и Y - действия этой группы на множестве S , тогда функция $f : X \rightarrow Y$ - эквивариантная, если $f(gx) = g f(x)$ для всех $g \in G$ и $x \in X$. Если, кроме того, f биективно, то f - эквивариантный изоморфизм.

Геометрический смысл определения эквивалентности заключается в следующем: если для CDP ψ прикрепить точки $P_i \in \mathbb{P}^1$ - получится дивизориальный многогранник на \mathbb{P}^1 , который соответствует рациональному

поляризованному Т-многообразию сложности 1. Эквивалентные CDP задают эквивариантно изоморфные Т-многообразия сложности 1 при условии "правильного" выбора точек P_i .

Определение 2.9 (Грань высоты 1). Грань F многогранника $P \in (M \times \mathbb{Z}) \otimes \mathbb{R}$ является гранью высоты 1, если существует вектор $u \in M^* \times \mathbb{Z}$ такой что $\langle v, u \rangle = 1$ для всех $v \in F$

Определение 2.10 (Свойство Фано). CDP обладает свойством Фано, если он эквивалентен какому-то CDP с базой \square и функциями ψ_1, \dots, ψ_n для которого существуют коэффициенты a_1, \dots, a_n такие что

1. $0 \in \square^\circ$
2. $\sum_{i=1}^n a_i = -2$
3. Для всех i $\psi_i(0) + a_i + 1 > 0$, и каждая грань $\Gamma(\psi_i + a_i + 1)$ - грань высоты 1
4. Для каждой грани F многогранника \square не являющейся гранью высоты 1 верно $\sum_{i=1}^n \psi_i \equiv 0$ на всей F

Все свойства определения Фано сохраняются для эквивалентных CDP при условии что трансформация базы оставляет начало координат внутри базы ([2]). Внутри класса рациональных поляризованных Т-многообразий сложности 1, CDP, удовлетворяющие свойству Фано, соответствуют каноническим Т-многообразиям Горенштейна-Фано с антиканонической поляризацией.

2.4 Генерация CDP из многогранников

Рассмотрим многогранник $P \in (M \times \mathbb{Z}) \otimes \mathbb{R}$ с вершинами на решетке $M \times \mathbb{Z}$. Из него можно получить CDP. Пусть π_1 - проекция на $M \otimes \mathbb{R}$, π_2 - проекция на \mathbb{R} . Возьмем $\square = \pi_1(P)$ и $\psi_1, \psi_2 : \square \rightarrow \mathbb{R}$: $\psi_1(u) = \max(\pi_2(\pi_1^{-1}(u) \cap P))$, $\psi_2(u) = -\min(\pi_2(\pi_1^{-1}(u) \cap P))$. Таким образом, из любого многогранника на решетке можно получить CDP с двумя функциями. И наоборот, из любого CDP с двумя функциями можно получить многогранник на решетке.

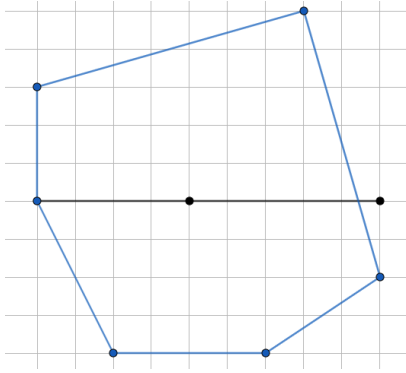


Рис. 1: Многогранник

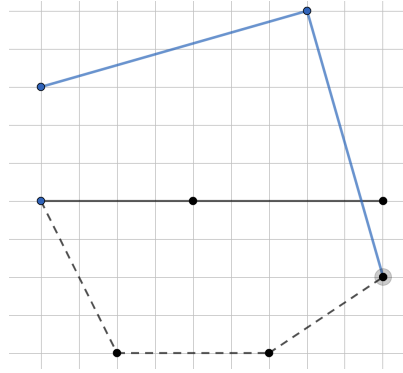


Рис. 2: $\square = \pi_1(P)$

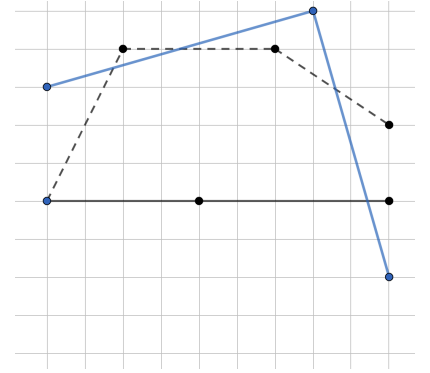


Рис. 3: CDP

Рис. 4: Пример (На втором и третьем рисунках пунктирная линия - грани, которые вошли в график ψ_1 , голубая линия - грани, которые вошли в график ψ_2)

Определение 2.11 (Торический CDP). CDP называется торическим, если он эквивалентен CDP с ровно двумя функциями

Любой торический CDP можно получить из многогранника с помощью алгоритма, описанного выше.

Торический CDP является Фано тогда и только тогда, когда соответствующий ему многогранник изоморфен рефлексивному.

3 Компьютерная реализация CDP

Весь код реализован на Python3 и представлен в Приложении, а также доступен по ссылке . В реализации используется класс Polyhedron из библиотеки sagemath.

Класс CDP имеет два поля - base (класс Polyhedron из sage) и psi_list - список объектов класса PiecewiseAffineFunction. Класс PiecewiseAffineFunction - реализация кусочно-линейной функции, класс имеет одно поле - affine_pieces, список объектов класса AffineFunction. Класс AffineFunction - реализация линейной функции, содержит два поля - domain (класс Polyhedron из sage, это область, на которой задана функция) и coeffs - список коэффициентов в формате $[a_0, a_1, \dots, a_{k-1}]$, тогда функция имеет вид $x_k = a_0 + a_1x_1 + \dots + a_{k-1}x_{k-1}$

CDP - сложный объект, нужно соблюсти множество условий, чтобы сконструировать правильный CDP, поэтому в конструкторе проверяется

корректность поданных на вход данных.

Пример корректного CDP:

Пример взят из статьи [2]

```
1 base = Polyhedron(vertices=[[-1], [1]])
2 # y = 1 + x, x \in [-1, 0]
3 f_11 = AffineFunction([1, 1], Polyhedron(vertices=[[-1], [0]]))
4 # y = 1 - x, x \in [0, 1]
5 f_12 = AffineFunction([1, -1], Polyhedron(vertices=[[0], [1]]))
6 f_1 = PiecewiseAffineFunction([f_11, f_12])
7 # y = 1/2x + 1/2, x \in [-1, 1]
8 f_2 = PiecewiseAffineFunction([AffineFunction([1 / 2, 1 / 2], Polyhedron(
    vertices=[[-1], [1]]))])
9 cdp = CDP([f_1, f_2], base)
10 print(cdp)
```

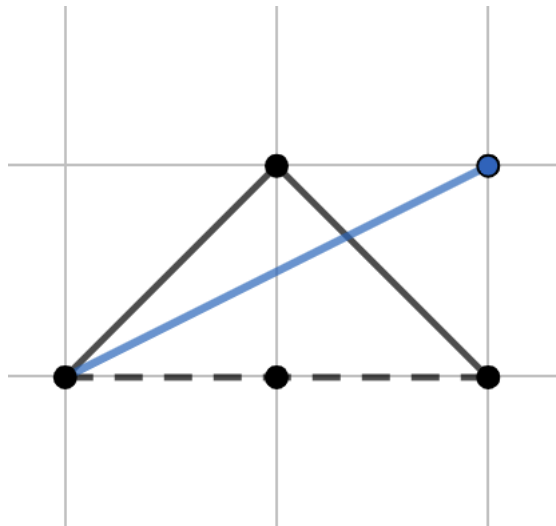


Рис. 5: Корректный CDP

Программа выведет:

```
CDP object , psi list :
Piecewise affine function :
Affine function 1 + x_1 with domain [(-1), (0)]
Affine function 1 - x_1 with domain [(0), (1)]

Piecewise affine function :
Affine function 0.5 + 0.5x_1 with domain [(-1), (1)],

base: (A vertex at (-1), A vertex at (1))
```

Пример некорректного CDP (сумма ψ_i отрицательная в одной из точек базы):

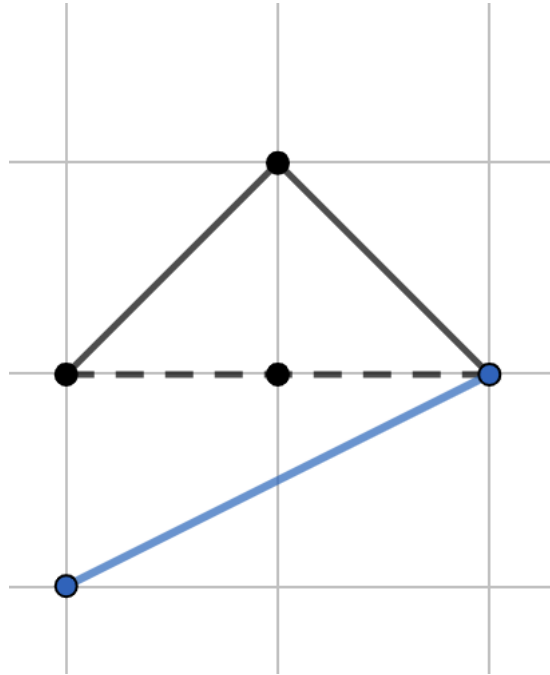


Рис. 6: Некорректный CDP

```

1 base = Polyhedron(vertices=[[-1], [1]])
2 # y = x, x \in [-1, 0]
3 f_11 = AffineFunction([0, 1], Polyhedron(vertices=[[-1], [0]]))
4 # y = -x, x \in [0, 1]
5 f_12 = AffineFunction([0, -1], Polyhedron(vertices=[[0], [1]]))
6 f_1 = PiecewiseAffineFunction([f_11, f_12])
7 # y = -1/2 + 1/2x, x \in [-1, 1]
8 f_2 = PiecewiseAffineFunction([AffineFunction([-1 / 2, 1 / 2], Polyhedron(
    vertices=[[-1], [1]]))])
9 cdp = CDP([f_1, f_2], base)

```

Программа выведет:

```

Traceback (most recent call last):
  File "test_cdp_validity.py", line 65, in <module>
    test.test_not_valid_cdp()
  File "test_cdp_validity.py", line 30, in
    test_not_valid_cdp
    cdp = CDP([f_1, f_2], base)
  File "cdp.py", line 27, in __init__
    raise ValueError(
ValueError: Not a valid CDP – sum of psi is -2.0 on A
vertex at (-1)

```


3.1 Преобразования CDP

Применим скашивание к CDP из примера 3: с коэффициентом -1 для первой функции и 1 для второй, с вектором (-1).

```
1 cdp = CDP([f_1, f_2], base)
2 cdp.shear([-1, 1], [-1])
3 print(cdp)
```

Программа выведет:

```
CDP object , psi list :
Piecewise affine function :
Affine function  $1 + 2x_1$  with domain  $[(-1), (0)]$ 
Affine function  $1$  with domain  $[(0), (1)]$ 

Piecewise affine function :
Affine function  $0.5 - 0.5x_1$  with domain  $[(-1), (1)]$ ,

base: (A vertex at  $(-1)$ , A vertex at  $(1)$ )
```

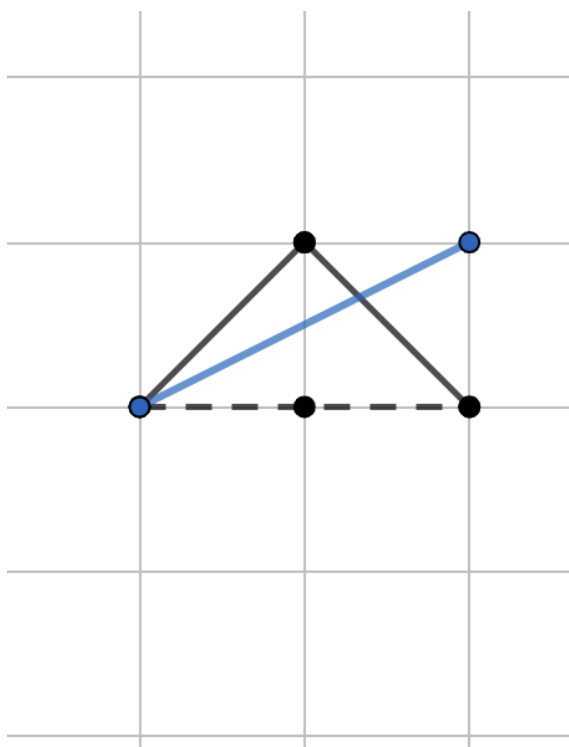


Рис. 7: Исходный CDP

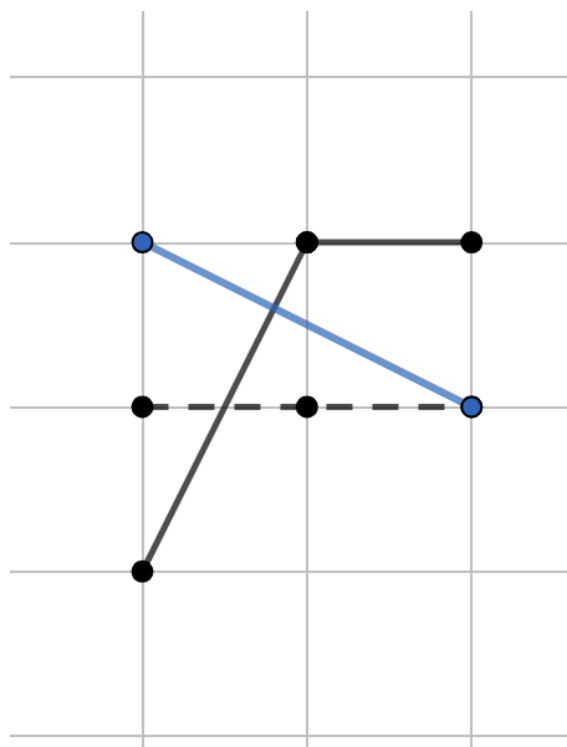


Рис. 8: После скашивания

Применим сдвиг с коэффициентами 1, -1 к результату предыдущего преобразования:

```
1 cdp.translate([1, -1])
2 print(cdp)
```

Результат:

```
CDP object , psi list :
Piecewise affine function :
Affine function  $2 + 2x_1$  with domain  $[(-1), (0)]$ 
Affine function  $2$  with domain  $[(0), (1)]$ 

Piecewise affine function :
Affine function  $-0.5 - 0.5x_1$  with domain  $[(-1), (1)]$ ,

base: (A vertex at  $(-1)$ , A vertex at  $(1)$ )
```

Применим трансформацию базы - зеркальное отображение относительно начала координат - к результату предыдущего преобразования:

```
1 A = matrix(ZZ, [[-1]])
2 phi = linear_transformation(A)
3 cdp.transform_base(phi)
4 print(cdp)
```

Результат:

```
CDP object , psi list :
Piecewise affine function :
Affine function  $2 - 2x_1$  with domain  $[(0), (1)]$ 
Affine function  $2$  with domain  $[(-1), (0)]$ 

Piecewise affine function :
Affine function  $-0.5 + 0.5x_1$  with domain  $[(-1), (1)]$ ,

base: (A vertex at  $(-1)$ , A vertex at  $(1)$ )
```

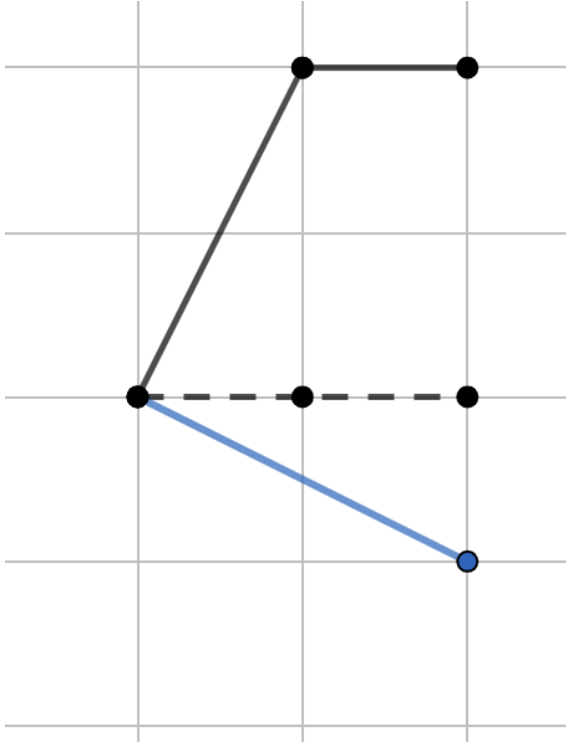


Рис. 9: После переноса

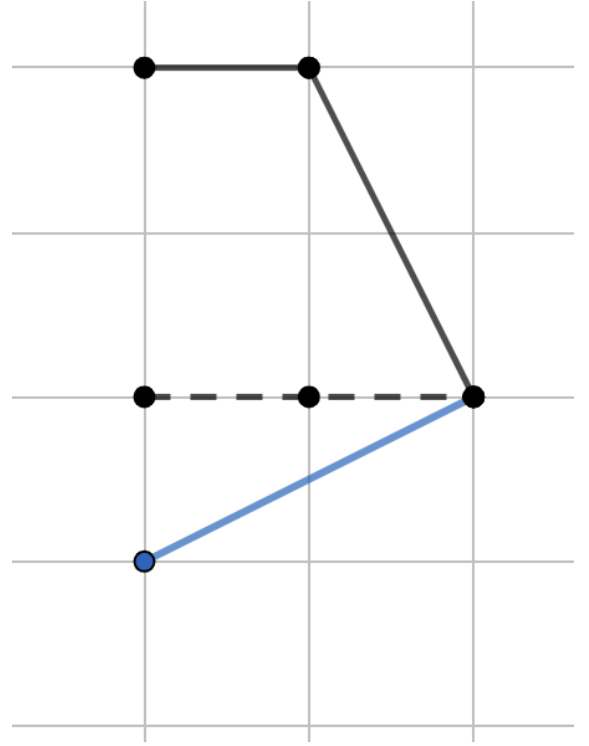


Рис. 10: После трансформации

3.2 Алгоритм проверки эквивалентности

Считаем, что трансформация базы всегда происходит только 1 раз, и это первая операция в цепочке преобразований (ниже будет доказано, почему такое предположение корректно).

Введем операцию обобщенного скашивания:

Определение 3.1 (Обобщенное скашивание). пусть $\sum_{i=1}^n v_i = 0$, тогда CDP с базой \square и набором функций $u \mapsto \psi_i(u) + \langle u, v_i \rangle$ получен из CDP $(\square, \psi_1, \dots, \psi_n)$ с помощью обобщенного скашивания.

Полный код алгоритма находится в Приложении в разделе 24

3.2.1 Описание

Пусть $CDP_1 = (\square, (\psi_1, \dots, \psi_n))$, $CDP_2 = (\square', (\psi'_1, \dots, \psi'_n))$, проверим их на эквивалентность:

- 1 Выберем нумерацию вершин в \square' . Если уже перебрали все возможные нумерации - завершаем алгоритм, CDP не эквивалентны.
- 2 Найдем такую матрицу A , которая переводит \square в \square' с учетом нумерации вершин, найдем матрицу обратного преобразования - A^{-1} .

Если не удалось найти A или A^{-1} - переходим к шагу 1 (выбираем другую нумерацию вершин). Применим A^{-1} к ψ_1, \dots, ψ_n , получим $\chi_1 := \psi_1 \phi^{-1}, \dots, \chi_n := \psi_n \phi^{-1}$.

- 3 Для каждой χ_i найдем множество $S_i = \{j : \chi_i \sim \psi'_j\}$ - множество индексов функций ψ'_j , имеющих такие же области линейности (иллюстрация 11). С помощью переноса или скашивания нельзя из линейной функции получить кусочно-линейную и наоборот, поэтому если для каких-то χ_i или ψ'_j не нашлось соответствия - возвращаемся на шаг 1, выбираем другую нумерацию вершин базы.

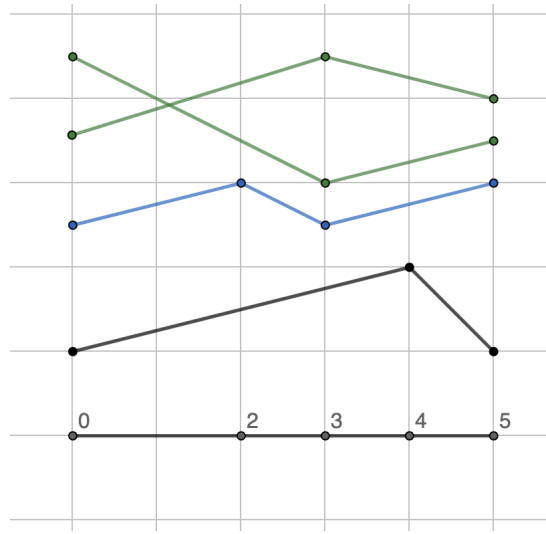


Рис. 11: Зеленые функции имеют области линейности $[[0, 3], [3, 5]]$, голубая - $[[0, 2], [2, 3], [3, 5]]$, черная - $[[0, 4], [4, 5]]$. Зеленые функции имеют одинаковые области линейности.

- 4 Каждая функция представляется в виде набора коэффициентов a_0, a_1, \dots, a_m , где a_0 - свободный коэффициент (константа), а a_1, \dots, a_m - коэффициенты перед переменными x_1, \dots, x_m . Выберем σ - перестановку функций χ_i с учетом классов эквивалентности (то есть $\sigma(i) \in S_i$ - $\chi_{\sigma(i)}$ и ψ'_i имеют одинаковые области линейности). Если все перестановки уже перебрали - переходим к шагу 1. Пусть у $\chi_{\sigma(i)}$ коэффициенты $a_{i0}, a_{i1}, \dots, a_{im}$, у ψ'_i - $b_{i0}, b_{i1}, \dots, b_{im}$, тогда если $\sum_{i=1}^n (b_{i0} - a_{i0}) = 0$, то существует перенос, позволяющий коэффициенты a_{i0} перевести в b_{i0} (перенос по определению меняет только свободные коэффициенты). Если $\sum_{i=1}^n (b_{i0} - a_{i0}) \neq 0$ - обозначаем

функции с коэффициентами $b_{i0}, a_{i1}, \dots, a_{im}$ как χ'_i и переходим к шагу 5, если нет - повторяем шаг 4 для другой перестановки.

- 5 Для каждой пары функций χ'_i, ψ'_i пытаемся найти вектор v_i , такой что $\chi'_i(u) + \langle u, v_i \rangle = \psi'_i(u)$, при этом для всех областей линейности должен получиться одинаковый вектор. Если для всех пар функций получилось найти такой вектор и $\sum_{i=1}^n v_i = 0$ - CDP эквивалентны (поскольку удалось найти последовательность из трансформации базы, переноса и обобщенного скашивания, переводящую CDP1 в CDP2), если нет - возвращаемся к пункту 4, выбирая другую перестановку.

3.2.2 Примеры работы:

Пример из статьи [2]. В предыдущем разделе (преобразования CDP) CDP2 был получен из CDP1 с помощью цепочки преобразований.

```

1 base1 = Polyhedron(vertices=[[-1], [1]])
2 # y = 1 + x, x \in [-1, 0]
3 f_11 = AffineFunction([1, 1], Polyhedron(vertices=[[-1], [0]]))
4 # y = 1 - x, x \in [0, 1]
5 f_12 = AffineFunction([1, -1], Polyhedron(vertices=[[0], [1]]))
6 f_1 = PiecewiseAffineFunction([f_11, f_12])
7 # y = 1/2 + 1/2x, x \in [-1, 1]
8 f_2 = PiecewiseAffineFunction([AffineFunction([1 / 2, 1 / 2], Polyhedron(
    vertices=[[-1], [1]]))])
9 cdp1 = CDP([f_1, f_2], base1)
10 base2 = Polyhedron(vertices=[[-1], [1]])
11 # y = 2, x \in [-1, 0]
12 y_11 = AffineFunction([2, 0], Polyhedron(vertices=[[-1], [0]]))
13 # y = 2 - 2x, x \in [0, 1]
14 y_12 = AffineFunction([2, -2], Polyhedron(vertices=[[0], [1]]))
15 y_1 = PiecewiseAffineFunction([y_11, y_12])
16 # y = -1/2 + 1/2x, x \in [-1, 1]
17 y_2 = PiecewiseAffineFunction([AffineFunction([-1 / 2, 1 / 2], Polyhedron(
    vertices=[[-1], [1]]))])
18 cdp2 = CDP([y_1, y_2], base2)
19 print(cdp1.equal(cdp2))
20 print(cdp2.equal(cdp1))

```

True

True

Пример для большей размерности:

```

1 base1 = Polyhedron(vertices=[[1, 0], [0, -1], [-1, 0], [0, 1]])

```

```

2 f_11 = AffineFunction([1, -1, -1], Polyhedron(vertices=[[0, 0], [1, 0], [0,
1]]))
3 f_12 = AffineFunction([1, -1, 1], Polyhedron(vertices=[[0, 0], [1, 0], [0,
-1]]))
4 f_13 = AffineFunction([1, 1, -1], Polyhedron(vertices=[[0, 0], [-1, 0], [0,
1]]))
5 f_14 = AffineFunction([1, 1, 1], Polyhedron(vertices=[[0, 0], [-1, 0], [0,
-1]]))
6 f_1 = PiecewiseAffineFunction([f_11, f_12, f_13, f_14])
7 f_21 = AffineFunction([1, 0, 0], Polyhedron(vertices=[[0, 1], [1, 0], [0,
-1]]))
8 f_22 = AffineFunction([1, 1, 1], Polyhedron(vertices=[[0, 0], [0, -1], [-1,
0]]))
9 f_23 = AffineFunction([1, 1, -1], Polyhedron(vertices=[[0, 0], [-1, 0], [0,
1]]))
10 f_2 = PiecewiseAffineFunction([f_21, f_22, f_23])
11 cdp1 = CDP([f_1, f_2], base1)
12 cdp2 = deepcopy(cdp1)
13 cdp2.shear([1, -1], [1, 1])
14 phi = linear_transformation(matrix(ZZ, [[-1, 0], [0, -1]]))
15 cdp2.transform_base(phi)
16 print(cdp1.equal(cdp2))
17 print(cdp2.equal(cdp1))

```

True

True

Пример не эквивалентных CDP:

```

1 base1 = Polyhedron(vertices=[[-1], [1]])
2 # y = 1 + x, x \in [-1, 0]
3 f_11 = AffineFunction([1, 1], Polyhedron(vertices=[[-1], [0]]))
4 # y = 1 - x, x \in [0, 1]
5 f_12 = AffineFunction([1, -1], Polyhedron(vertices=[[0], [1]]))
6 f_1 = PiecewiseAffineFunction([f_11, f_12])
7 # y = 1/2 + 1/2x, x \in [-1, 1]
8 f_2 = PiecewiseAffineFunction([AffineFunction([1 / 2, 1 / 2], Polyhedron(
vertices=[[-1], [1]]))])
9 cdp1 = CDP([f_1, f_2], base1)
10 base2 = Polyhedron(vertices=[[-1], [1]])
11 # y = 2, x \in [-1, 0]
12 y_11 = AffineFunction([2, 0], Polyhedron(vertices=[[-1], [0]]))
13 # y = 2 - x, x \in [0, 1]
14 y_12 = AffineFunction([2, -1], Polyhedron(vertices=[[0], [1]]))
15 y_1 = PiecewiseAffineFunction([y_11, y_12])
16 # y = -1/2 + 1/2x, x \in [-1, 1]
17 y_2 = PiecewiseAffineFunction([AffineFunction([-1 / 2, 1 / 2], Polyhedron(
vertices=[[-1], [1]]))])
18 cdp2 = CDP([y_1, y_2], base2)

```

```

19 print ( cdp1 . equal ( cdp2 ) )
20 print ( cdp2 . equal ( cdp1 ) )

```

```

False
False

```

3.2.3 Доказательство корректности

Теорема 3.1. Любую цепочку преобразований CDP можно представить в виде трех последовательных преобразований: трансформации базы, переноса и обобщенного скашивания.

Доказательство. Все трансформации базы, присутствующие в цепочке преобразований, можно перенести в ее начало по Лемме 3.3. После этого все трансформации можно объединить в одну по Лемме 3.4. Далее можно поменять местами переносы и скашивания так, чтобы все переносы происходили последовательно следом за трансформацией базы по Лемме 3.5. Оставшиеся операции скашивания можно объединить в одно обобщенное скашивание по Лемме 3.7. \square

Теорема 3.2. Если удалось найти цепочку преобразований, переводящую CDP1 в CDP2 и состоящую из трех последовательных преобразований - трансформации базы, переноса и обобщенного скашивания - то эти CDP эквивалентны.

Доказательство. Нужно доказать, что существует цепочка, состоящая из трансформаций базы, переносов и обычных скашиваний, переводящая CDP1 в CDP2, то есть нужно доказать, что обобщенное скашивание можно представить в виде композиции обычных. Это доказывается в лемме 3.8. \square

Лемма 3.3. Трансформацию базы можно всегда считать первой в цепочке преобразований

Доказательство. Пусть $CDP_1 (\square, \psi_1, \dots, \psi_n)$ переводится в $CDP_2 (\square', \psi'_1, \dots, \psi'_n)$ с помощью цепочки преобразований BC , где B - операция скашивания, C - трансформация базы. Покажем, что тогда существует другая цепочка преобразований - $C'B'$, где C' - трансформация

базы, B' - операция скашивания, - которая также переводит CDP_1 в CDP_2 .

$\psi_i(u) \xrightarrow{C'B'} \psi'_i(u_{new}) = \psi_i(\phi^{-1}(u_{new})) + \beta_i\langle u_{new}, v \rangle$
 $\psi_i(u) \xrightarrow{B} \psi_i(u) + \beta_i\langle u, v_1 \rangle$, положим $v_1 = \phi(v)$, тогда $\psi_i(u) \xrightarrow{B} \psi_i(u) + \beta_i\langle u, \phi(v) \rangle$, теперь применим трансформацию C : $\psi_i(u) \xrightarrow{BC} \psi_i(\phi^{-1}(u_{new})) + \beta_i\langle \phi^{-1}(u_{new}), \phi(v) \rangle = \psi_i(\phi^{-1}(u_{new})) + \beta_i\langle u_{new}, v \rangle = \psi'_i(u_{new})$, то есть цепочки преобразований BC и $C'B'$ дают одинаковый результат

Аналогично для операции переноса (translation) \square

Лемма 3.4. Можно считать, что трансформация базы происходит только 1 раз в цепочке преобразований

Доказательство. Все трансформации базы можно перенести в начало цепочки по предыдущему пункту, а композиция линейных обратимых преобразований также является линейным обратимым преобразованием, поэтому можно заменить композицию трансформаций на единственную трансформацию базы. \square

Лемма 3.5. Перенос и скашивание можно менять местами

Доказательство. $\psi_i(u) + \beta_i\langle u, v \rangle + \alpha_i = \psi_i(u) + \alpha_i + \beta_i\langle u, v \rangle$ \square

Лемма 3.6. Два переноса можно объединить в один

Доказательство. $\psi_i + \alpha_i + \alpha'_i$, $\sum_{i=1}^n \alpha_i = 0$, $\sum_{i=1}^n \alpha'_i = 0$ - значит и $\sum_{i=1}^n (\alpha_i + \alpha'_i) = 0$, тогда $\psi_i + \alpha''_i$, $\alpha''_i = \alpha_i + \alpha'_i$ - корректный перенос. \square

Лемма 3.7. Два скашивания можно объединить в одно обобщенное скашивание

Доказательство. Пусть $\phi_i(u) = \psi_i(u) + \beta_i\langle u, v \rangle$, $\chi_i(u) = \phi_i(u) + \beta'_i\langle u, v' \rangle$, тогда $\chi_i(u) = \psi_i(u) + \beta_i\langle u, v \rangle + \beta'_i\langle u, v' \rangle = \psi_i(u) + \langle u, w \rangle \sum_{i=1}^n \beta_i = 0$, $\sum_{i=1}^n \beta'_i = 0$, тогда $\sum_{i=1}^n \beta_i\langle u, v \rangle = 0$ и $\sum_{i=1}^n \beta'_i\langle u, v' \rangle = 0$, а значит $\sum_{i=1}^n w = 0$, то есть χ_i можно получить из ϕ_i с помощью операции обобщенного скашивания. \square

Лемма 3.8. Любое обобщенное скашивание можно представить в виде суммы обычных

Доказательство. Пусть v_1, \dots, v_n - обобщенное скашивание. Покажем, что v_i можно представить в виде $\beta_{i1}v'_1 + \dots + \beta_{ik}v'_k, \forall j \sum_{i=1}^n \beta_{ij} = 0$

$$v_1 = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix}, \dots, v_n = \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} = a_{11} \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + a_{21} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \dots + a_{m1} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} = a_{12} \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + a_{22} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \dots + a_{m2} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

\vdots

$$\begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} = a_{1n} \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + a_{2n} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \dots + a_{mn} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$\forall i \sum_{j=1}^n a_{ij} = 0$ (суммы коэффициентов в столбцах), поскольку $\sum_{i=1}^n v_i = 0$.

Значит, $a_{1j} \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, a_{2j} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \dots, a_{mj} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$ - корректные операции скашивания

(в качестве вектора v для каждого преобразования берем базисный вектор,

а в качестве коэффициентов β_i - координаты исходных векторов).

Таким образом, если в пункте 4 нашли обобщенное скачивание, которое переводит CDP_1 в CDP_2 - найдется и последовательность обычных, которые делают то же самое. \square

3.3 Алгоритм генерации CDP из многогранников

Полный код алгоритма находится в Приложении в разделе 30 Алгоритм принимает на вход многогранник P размерности k .

1. Для каждой грани находим вектор нормали, внешний по отношению к многограннику. Если проекция вектора нормали на последнюю ось координат (x_k) положительная - относим грань к списку *psi1_list*, если отрицательная - к *psi2_list*, а если равна нулю - пропускаем грань, не относим ее ни к какому списку.
2. Находим проекцию многогранника на подпространство (x_1, \dots, x_{k-1}) (для этого достаточно избавиться от последней координаты каждой из вершин и взять выпуклую оболочку полученного множества точек) - это база CDP.
3. Каждая грань соответствует области линейности одной из функций ψ_1, ψ_2 (класс AffineFunction). Для каждой грани из списка *psi1_list* выполняем следующее: по точкам, соответствующим вершинам грани, строим уравнение гиперплоскости. Таким образом заполняем поле *coefs* класса AffineFunction. Берем выпуклую оболочку проекций вершин грани на подпространство (x_1, \dots, x_{k-1}) - получается многогранник, на котором функция ψ_1 линейна. Это соответствует полю *domain* класса AffineFunction. Аналогично для списка *psi2_list* (для ψ_2 берем все коэффициенты с минусом, поскольку нужно получить зеркальное отображение относительно (x_1, \dots, x_{k-1})).

3.3.1 Примеры работы алгоритма

Генерация CDP из пирамиды:

```

1 poly = Polyhedron(vertices=[[1, 0, 0], [0, 1, 0], [0, 0, 1], [-1, 0, 0],
2                               [0, 0, -1]])

```

```

3 cdp = generate_cdp_from_polytope(poly)
4 print(cdp)

```

Результат:

```

CDP object , psi list :
Piecewise affine function :
Affine function  $1 - x_1 - x_2$  with domain  $[(0, 0), (0, 1), (1, 0)]$ 
Affine function  $1 + x_1 - x_2$  with domain  $[(-1, 0), (0, 0), (0, 1)]$ 

Piecewise affine function :
Affine function  $1 - x_1 - x_2$  with domain  $[(0, 0), (0, 1), (1, 0)]$ 
Affine function  $1 + x_1 - x_2$  with domain  $[(-1, 0), (0, 0), (0, 1)]$ ,

base: (A vertex at  $(-1, 0)$ , A vertex at  $(1, 0)$ , A vertex at  $(0, 1)$ )

```

Генерация CDP для двумерного случая:

```

1 poly = Polyhedron(vertices=[[-2, 0], [0, 2], [1, 2], [2, 1], [2, -2], [-2, -2]])
2 cdp = generate_cdp_from_polytope(poly)
3 print(cdp)

```

Результат:

```

CDP object , psi list :
Piecewise affine function :
Affine function  $3 - x_1$  with domain  $[(1), (2)]$ 
Affine function  $2$  with domain  $[(0), (1)]$ 
Affine function  $2 + x_1$  with domain  $[(-2), (0)]$ 

Piecewise affine function :
Affine function  $2$  with domain  $[(-2), (2)]$ ,

base: (A vertex at  $(-2)$ , A vertex at  $(2)$ )

```

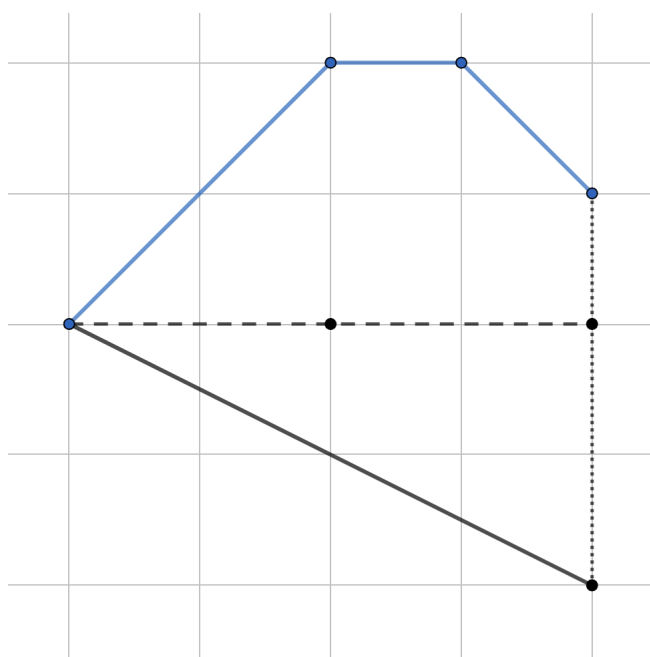


Рис. 12: Пример генерации CDP из многогранника: голубым обозначены ребра, которые войдут в ψ_1 , черным (сплошным) - ребра, которые войдут в ψ_2

4 Приложение

4.1 piecewise_affine_function.py

```
1 #!/usr/bin/env sage
2 from typing import List
3 from sage.all import *
4 import numpy as np
5
6
7 class AffineFunction:
8     def __init__(self, coefficients: List[float], domain: Polyhedron):
9         dim = len(domain.vertices()[0].vector())
10        if len(coefficients) != dim + 1:
11            raise ValueError(f'Domain dimension {len(coefficients) - 1} '
12                             f'and coefficients list dimension {dim} do not match')
13        self.coefs = list(coefficients)
14        self.domain = domain
15        self.dim = dim
16
17    def __eq__(self, other):
18        return self.coefs == other.coefs and self.domain == other.domain
19
20    def value(self, x: List[int]):
21        if x not in self.domain:
22            raise ValueError(
23                f'{x} is not in function domain {self.domain.vertices()}')
24        if len(x) != self.dim:
25            raise ValueError(f'Wrong dimension of x: {len(x)}')
26        return sum([a * b for a, b in zip(self.coefs[1:], x)]) + self.coefs[0]
27
28    def _coef_str(self, i):
29        if self.coefs[i] == 0:
30            return ''
31        if self.coefs[i] == 1:
32            return f'+ x_{i}'
33        elif self.coefs[i] == -1:
34            return f'- x_{i}'
35        elif self.coefs[i] > 0:
36            return f'+ {self.coefs[i]}x_{i}'
37        return f'- {-self.coefs[i]}x_{i}'
38
39    def __str__(self):
40        coefs_repr = []
41        if self.coefs[0] != 0 or len(self.coefs) == 2 and self.coefs[1] == 0:
```

```

42         coefs_repr.append(str(self.coefs[0]))
43     for i in range(1, len(self.coefs)):
44         coefs_repr.append(self._coef_str(i))
45     coefs_repr = "".join(coefs_repr)
46     return f'Affine function {coefs_repr} with domain ' \
47           f'[{vert.vector() for vert in self.domain.vertices()}]'
48
49
50 class PiecewiseAffineFunction:
51     def __init__(self, affine_pieces: List[AffineFunction]):
52         self.affine_pieces = affine_pieces
53
54     def __eq__(self, other):
55         if not len(self.affine_pieces) == len(other.affine_pieces):
56             return False
57         for piece in self.affine_pieces:
58             if not piece in other.affine_pieces:
59                 return False
60         return True
61
62     def value(self, x: List[int]):
63         for piece in self.affine_pieces:
64             if x in piece.domain:
65                 return piece.value(x)
66         raise ValueError(f'{x} is not in function domain')
67
68     def transform(self, phi, phi_inverse):
69         for j in range(len(self.affine_pieces)):
70             res = np.matmul(self.affine_pieces[j].coefs[1:], phi_inverse)
71             self.affine_pieces[j].coefs = [self.affine_pieces[j].coefs[0]]
72             self.affine_pieces[j].coefs.extend(res)
73             vertices = []
74             for vert in self.affine_pieces[j].domain.vertices():
75                 vertices.append(phi(vert.vector()))
76             self.affine_pieces[j].domain = Polyhedron(vertices=vertices)
77
78     def _domains_mapping(self, other_psi):
79         if len(self.affine_pieces) != len(other_psi.affine_pieces):
80             return False, 0
81         domains_mapping = [0 for _ in range(len(other_psi.affine_pieces))]
82         for i, piece in enumerate(self.affine_pieces):
83             for j, other_piece in enumerate(other_psi.affine_pieces):
84                 if piece.domain == other_piece.domain:
85                     domains_mapping[i] = j
86         return domains_mapping

```

```

87
88 def can_be_translated(self, other_psi):
89     domains_mapping = self._domains_mapping(other_psi)
90     alpha = other_psi.affine_pieces[domains_mapping[0]
91                                     ].coefs[0] - self.affine_pieces[0].coefs[0]
92     for i in range(1, len(domains_mapping)):
93         a = other_psi.affine_pieces[domains_mapping[i]
94                                     ].coefs[0] - self.affine_pieces[i].coefs[0]
95         if a != alpha:
96             return False, 0
97     return True, alpha
98
99 def cat_be_sheared(self, other_psi):
100     domains_mapping = self._domains_mapping(other_psi)
101     n = len(other_psi.affine_pieces[domains_mapping[0]].coefs)
102     v = [other_psi.affine_pieces[domains_mapping[0]].coefs[i] -
103          self.affine_pieces[0].coefs[i]
104          for i in range(1, n)]
105     for j in range(1, len(domains_mapping)):
106         other_v = [other_psi.affine_pieces[domains_mapping[j]].coefs[i] -
107                   self.affine_pieces[j].coefs[i]
108                   for i in range(1, n)]
109         if other_v != v:
110             return False
111     return True
112
113 def __str__(self):
114     resp = '\n'.join([str(piece) for piece in self.affine_pieces])
115     return 'Piecewise affine function:\n' + resp

```

4.2 cdp.py

```
1 #!/usr/bin/env sage
2 from sage.all import *
3 from typing import List
4 from piecewise_affine_function import PiecewiseAffineFunction
5 from collections import defaultdict
6 from itertools import permutations
7 import numpy as np
8
9
10 class CDP:
11     def __init__(
12         self, psi_list: List[PiecewiseAffineFunction], base: Polyhedron):
13         # Check that sum of psi is non negative on borders of domains
14         # (check polytop vertices of domains laying inside of function scope)
15         for psi in psi_list:
16             for piece in psi.affine_pieces:
17                 for vert in piece.domain.vertices():
18                     if vert in base:
19                         try:
20                             s = sum([ps.value(vert.vector())
21                                     for ps in psi_list])
22                         except ValueError:
23                             continue
24                         else:
25                             if s < 0:
26                                 raise ValueError(
27                                     f'Not a valid CDP - sum of psi is {s} on {vert}'
28                                     ')
29         # Check that sum of psi is non negative on base vertices
30         for vert in base.vertices():
31             try:
32                 s = sum([ps.value(vert) for ps in psi_list])
33             except ValueError:
34                 raise ValueError('Not a valid CDP: psi is not defined on base')
35             else:
36                 if s < 0:
37                     raise ValueError(
38                         f'Not a valid CDP - sum of psi is {s} on {vert}')
39         self.psi_list = psi_list
40         self.base = base
41         self.n = len(self.base.vertices()[0].vector())
42         self.k = len(self.base.vertices())
43         self.base_adjacency_map = defaultdict(set)
```



```

43
44 def __str__(self):
45     psi_str = "\n\n".join([str(psi) for psi in self.psi_list])
46     return f'CDP object, psi list:\n{psi_str},\n\nbase: {self.base.vertices()}\n'
47
48 def __eq__(self, other):
49     """
50     Note: the order of the functions in psi_list is not important for this
51     implementation.
52     """
53     if not self.base == other.base:
54         return False
55     if not len(self.psi_list) == len(other.psi_list):
56         return False
57     for psi in self.psi_list:
58         if psi not in other.psi_list:
59             return False
60     return True
61
62 def _build_adjacency_map(self):
63     for i in range(self.k):
64         for j in range(i + 1, self.k):
65             if self.base.vertices()[i].is_incident(
66                 self.base.vertices()[j]):
67                 self.base_adjacency_map[i].add(j)
68                 self.base_adjacency_map[j].add(i)
69
70 def transform_base(self, phi: linear_transformation):
71     vertices = []
72     for vert in self.base.vertices():
73         vertices.append(phi(vert.vector()))
74     self.base = Polyhedron(vertices=vertices)
75     try:
76         inv = phi.inverse().matrix()
77     except ZeroDivisionError:
78         raise ValueError(f'phi is not invertible')
79     inv = np.array([np.array(row) for row in inv])
80     for i in range(len(self.psi_list)):
81         self.psi_list[i].transform(phi, inv)
82
83 def translate(self, alpha_list: List[int]):
84     if len(alpha_list) != len(self.psi_list):
85         raise ValueError(f'Length of alpha_list is {len(alpha_list)}, '
86                             f'should be {len(self.psi_list)}')

```

```

86         if sum(alpha_list) != 0:
87             raise ValueError('Sum of coefficients should be 0')
88         for idx, psi in enumerate(self.psi_list):
89             for piece in psi.affine_pieces:
90                 piece.coefs[0] += alpha_list[idx]
91
92     def shear(self, beta_list: List[int], v: List[int]):
93         if len(beta_list) != len(self.psi_list):
94             raise ValueError(f'Length of beta_list is {len(beta_list)}, '
95                               f'should be {len(self.psi_list)}')
96         if sum(beta_list) != 0:
97             raise ValueError('Sum of coefficients should be 0')
98         m = len(self.psi_list[0].affine_pieces[0].coefs) - 1
99         if len(v) != m:
100             raise ValueError(f'Wrong dimension of v: {len(v)}, should be {m}')
101         for idx, psi in enumerate(self.psi_list):
102             for piece in psi.affine_pieces:
103                 for j, coef in enumerate(v):
104                     piece.coefs[j + 1] += coef * beta_list[idx]
105
106     def _vert_permutation_is_valid(self, perm):
107         """Check that vertices permutation saves incidence
108         """
109         for vert, inc_list in self.base_adjacency_map.items():
110             for other_vert in inc_list:
111                 if not perm[other_vert] in self.base_adjacency_map[perm[vert]]:
112                     return False
113         return True
114
115     def _get_transform_matrix(self, points, point_images):
116         left = np.matmul(points.T, points)
117         try:
118             inverse = np.linalg.inv(left)
119         except np.linalg.LinAlgError:
120             raise ValueError(f'Base of the CDP should be non-degenerate '
121                               f'(dimension {self.n} is provided, but '
122                               'the real dimension is lower)')
123         A = np.matmul(np.matmul(point_images, points), inverse)
124         return A
125
126     def _domains_match(self, psi1: PiecewiseAffineFunction,
127                        psi2: PiecewiseAffineFunction) -> bool:
128         if len(psi1.affine_pieces) != len(psi2.affine_pieces):
129             return False
130         return set([p.domain for p in psi1.affine_pieces]) == set(

```

```

131         [p.domain for p in psi2.affine_pieces])
132
133     def _get_equivalence_classes(self, other_psi_list):
134         used = [False for _ in range(len(other_psi_list))]
135         classes = [[] for _ in range(len(self.psi_list))]
136         for i, psi1 in enumerate(self.psi_list):
137             for j, psi2 in enumerate(other_psi_list):
138                 if self._domains_match(psi1, psi2):
139                     used[j] = True
140                     classes[i].append(j)
141         if len([c for c in classes if len(c) == 0]) > 0 or len(
142             [u for u in used if not u]) > 0:
143             return False, []
144         return True, classes
145
146     def _list_mappings(self, classes):
147         dicts_res = self._list_mappings_recursive(classes, dict(), idx=0)
148         res = []
149         for small_res in dicts_res:
150             res.append([0] * len(classes))
151             for k, v in small_res.items():
152                 res[-1][v] = k
153         return res
154
155     def _list_mappings_recursive(self, classes, current_dict, idx=0):
156         res = []
157         for elem in classes[idx]:
158             if elem in current_dict:
159                 continue
160             if idx + 1 == len(classes):
161                 d = deepcopy(current_dict)
162                 d[elem] = idx
163                 return [d]
164             d = deepcopy(current_dict)
165             d[elem] = idx
166             res += self._list_mappings_recursive(classes, d, idx + 1)
167         return res
168
169     def _can_be_translated(self, mapping, other_psi_list):
170         alpha_list = []
171         for i, j in enumerate(mapping):
172             can, alpha = self.psi_list[i].can_be_translated(other_psi_list[j])
173             if not can:
174                 return False
175             alpha_list.append(alpha)

```

```

176         if sum(alpha_list) != 0:
177             return False
178         return True
179
180     def _can_be_sheared(self, mapping, other_psi_list):
181         for i, j in enumerate(mapping):
182             if not self.psi_list[i].cat_be_sheared(other_psi_list[j]):
183                 return False
184         return True
185
186     def equal(self, other_cdp):
187         """
188         Suppose that zero functions are already removed
189         """
190         # Check that self.base is convertible to other_cdp.base with some phi
191         # Check that psi lists split into matching equivalence classes
192         # Check that a constant for translation exists
193         # Check that a vector for shearing exists
194         if len(self.base.vertices()) != len(other_cdp.base.vertices()):
195             return False
196         if len(self.psi_list) != len(other_cdp.psi_list):
197             return False
198         G = np.array([np.array(vert.vector())
199                       for vert in other_cdp.base.vertices()])
200         G = G.T
201         for perm in permutations([i for i in range(self.k)]):
202             if not self._vert_permutation_is_valid(perm):
203                 continue
204             V = np.array([np.array(self.base.vertices()[i].vector())
205                           for i in perm])
206             # A transforms base of one CDP to the base of another
207             A = self._get_transform_matrix(V, G)
208             A = linear_transformation(matrix(QQ, A))
209             cdp_after_base_transform = deepcopy(self)
210             try:
211                 cdp_after_base_transform.transform_base(A)
212             except ValueError:
213                 continue
214             splits, classes = cdp_after_base_transform._get_equivalence_classes(
215                 other_cdp.psi_list)
216             if not splits:
217                 continue
218             mappings = cdp_after_base_transform._list_mappings(classes)
219             for m in mappings:
220                 can = cdp_after_base_transform._can_be_translated(

```

```
221         m, other_cdp.psi_list)
222     if not can:
223         continue
224     can = cdp_after_base_transform._can_be_sheared(
225         m, other_cdp.psi_list)
226     if not can:
227         continue
228     return True
229 return False
```

4.3 generate_cdp.py

```

1  #!/usr/bin/env sage
2  from sage.all import *
3  from sympy.matrices import Matrix
4  from sympy import Rational, lcm
5  from piecewise_affine_function import PiecewiseAffineFunction, AffineFunction
6  from cdp import CDP
7
8
9  def generate_cdp_from_polytope(poly: Polyhedron):
10     # Walk over all facets, if facet's normal projection to x_n is positive -
11     # this facet is a part of psi_1 graph, otherwise - part of -psi_2 graph
12
13     def plane_from_points(points):
14         # (p_1l, ..., p_1n), ..., (p_nl, ..., p_nm)
15         # p_1l, p_2l, ..., p_nl
16         # ... = P
17         # p_1n, p_2n, ..., p_nm
18         # AP = b, b = (1, ..., 1) => A = bP^-1
19         # 1 - a_1*x_1 - ... - a_n*x_n = 0, return [1, -a1, ..., -a_n]
20         points = points[:len(points[0])]
21         b = Matrix([[1 for i in range(len(points[0]))]])
22         P = Matrix(points).T
23         A = b * P.inv()
24         return [Rational(1)] + [-a for a in list(A.row(0))]
25
26     def integer_coefs_from_rational(coefs):
27         qs = [c.q for c in coefs]
28         l = lcm(qs)
29         coefs = [c * l for c in coefs]
30         return [c / coefs[-1] for c in coefs]
31
32     def piecewise_from_facets(facets, invert_coefs=False):
33         pieces = []
34         for facet in facets:
35             # Find a plane equation with rational coefficients from facet's
36             vertices
37             # and multiply by least common multiple of denominators to obtain
38             integer
39             # coefficients.
40             r = integer_coefs_from_rational(
41                 plane_from_points([p.vector() for p in facet]))
42             if invert_coefs:
43                 r = [-c for c in r]

```

```

42         pieces.append(AffineFunction(coefficients=r[: -1], domain=Polyhedron(
43             vertices=[p.vector()[: -1] for p in facet]))
44     return PiecewiseAffineFunction(affine_pieces=pieces)
45
46     psi1_facets = []
47     psi2_facets = []
48     for facet in poly.facets():
49         # Find out whether a facet belongs to psi1 or psi2
50         coef = facet.normal_cone(direction='outer').rays_list()[0][ -1]
51         if coef > 0:
52             psi1_facets.append(facet.vertices())
53         elif coef < 0:
54             psi2_facets.append(facet.vertices())
55     psi1 = piecewise_from_facets(psi1_facets, invert_coefs=True)
56     psi2 = piecewise_from_facets(psi2_facets)
57     return CDP(psi_list=[psi1, psi2], base=Polyhedron(
58         vertices=[v.vector()[: -1] for v in poly.vertices()]))

```

Список литературы

- [1] Nathan Ilten и Hendrik Suss. K-stability for Fano manifolds with torus action of complexity one. 2017. DOI: <https://arxiv.org/pdf/1507.04442.pdf>.
- [2] Marni Mishna Nathan Ilten и Charlotte Trainor. Classifying Fano complexity-one T-varieties via Divisorial Polytopes. 2018. DOI: <https://arxiv.org/pdf/1710.04146.pdf>.
- [3] Nathan Owen Ilten Hendrik Süß. Polarized Complexity-One T-Varieties. 2011. DOI: <https://arxiv.org/pdf/0910.5919.pdf>.