,

Имплементация алгоритма Левина-Ливната-Цвика приближенного решения MAX 2-SAT с точностью 0.940

Сурин Родион Глебович
ФПМИ МФТИ

Аннотация

Задача $\mathsf{MAX2-SAT}$ состоит в том, чтобы в конъюнкции из дизъюнктов, состоящих из двух литералов, присвоить значения переменным таким образом, чтобы число выполненных дизъюнктов было максимальным. В данном проекте будет реализован алгоритм Левина-Ливната-Цвика [1] приближенного решения с точностью 0.940. В алгоритме каждому литералу сопоставляется вектор, строится задача полуопределенного программирования на основе этих векторов, ее решения поворачиваются определенным образом, а затем округляются с помощью случайных гиперплоскостей для получения итогового ответа.

1 Введение

1.1 Общая информация о задаче

Задача $\mathsf{MAX\ 2-SAT}$ является частным случаем задачи $\mathsf{MAX-SAT}$. Последняя задача состоит в том, чтобы в конъюнкции с произвольным числом дизъюнктов и произвольным числом литералов в самих дизъюнктах найти такие значения переменных, чтобы число выполненных дизъюнктов было максимальным. В $\mathsf{MAX\ 2-SAT}$ в каждом из дизъюнктов должно быть по 2 литерала. $\mathsf{MAX-SAT}$ сама по себе является обобщением задачи SAT, в которой нужно просто проверить, является ли формула выполнимой или нет.

Задача $\mathsf{MAX2-SAT}$ имеет различные применения, связанные с максимизацией числа желаемых условий. Например, может быть поставлена задача разделения некоторых предметов на 2 группы, и выдвинут ряд условий на пары предметов в терминах этого разделения. Тогда для преобразования ее к $\mathsf{MAX2-SAT}$ нужно каждому предмету сопоставить переменную, которая будет принимать значение 0, когда предмет в первой группе, и 1, когда во второй.

Известно, что $\mathsf{MAX2-SAT}$, $\mathsf{MAX-SAT}$ и SAT все являются **NP**-полными.

1.2 История задачи и значение результатов

Для задачи $\mathsf{MAX\ 2-SAT}$ есть тривиальная $\frac{3}{4}$-аппроксимация. Она заключается в случайном независимом присвоении каждой переменной равновероятно значения 1 или 0. Каждый из дизъюнктов будет выполнен в 3 из 4 случаев, поэтому математическое ожидание числа выполненных дизъюнктов равно $\frac{3}{4}$. История улучшенных аппроксимация является следующей:

1. В 1995 году был опубликован алгоритм Гёманса-Уильямсона [2], основанный на сведении $\mathsf{MAX2-SAT}$ к задаче полуопределенного программирования (Semidefinite Programming / SDP), дающий 0.878-аппроксимацию.

2. Также в 1995 в работе Фейге и Гёманса [3] был представлен поворот решений задачи SDP, дающий 0.931-аппроксимацию.

3. В 2001 были опубликованы работы Матууры и Мацуи [4], [5], по итогу которых было получено 0.935-приближение, основанное на использовании для итогового округления векторов случайной гиперплоскости, основанной на распределении, отличном от нормального.

4. Наконец, в 2002 году был опубликован алгоритм Левина-Ливната-Цвика [1], основанный на алгоритме Гёманса-Уильямсона [2] и совмещающий в себе улучшения из работ Фейге и Гёманса [3] и Матууры и Мацуи [4], [5].

2 Техническая часть

Задача $\mathsf{MAX2-SAT}$ с булевыми переменными $x_1, ..., x_n$ состоит из набора дизъюнктов $C_1, ..., C_m$. Каждый дизъюнкт C_i имеет вид $z_1 \vee z_2$, где каждая z_j является либо переменной x_k, либо ее отрицанием $\overline{x_k}$. Каждая такая z_j называется *литералом*. Задача заключается в том, чтобы присвоить переменным $x_1, ..., x_n$ булевы значения 0 или 1 так, чтобы число выполненных дизъюнктов было максимальным. Дизъюнкт является выполненным, если хотя бы один из литералов в нем имеет значение 1 (если переменной присвоено значение 0, то ее отрицанию присвоено 1, и наоборот).

Для удобства положим $x_{n+i} = \overline{x_i}$ для $1 \leq i \leq n$, а также положим $x_0 = 0$. Тогда каждый дизъюнкт будет вида $x_i \vee x_j$, где $0 \leq i, j \leq 2n$. В таком случае цель состоит в присвоении $x_1, ..., x_{2n}$ булевых значений так, что $x_{n+i} = \overline{x_i}$ и что число выполненных дизъюнктов максимально.

SDP-релаксация $\mathsf{MAX2-SAT}$, использованная в работе Фейге и Гёманса [3], приведена в **рис. 1**. В ней каждому литералу x_i, где $0 \leq i \leq 2n$, сопоставлен вектор v_i единичной длины. Так как $x_0 = 0$, вектор v_0 соответствует константе 0. Для согласованности потребуем $v_i \cdot v_{n+i} = -1$ для $1 \leq i \leq n$. Так как решение задачи SDP зависит только от скалярных произведений между векторами, не умаляя общности, можно считать, что $v_0 = (1, 0, ..., 0) \in \mathbb{R}^{n+1}$. Так как для каждых $1 \leq i, j \leq 2n$ верны "треугольные ограничения", а также так как $v_i = -v_{n+i}$ для $1 \leq i \leq n$, то если v_0, v_i, v_j присутствуют в решении задачи SDP, верно:

$$v_0 \cdot v_i + v_0 \cdot v_j + v_i \cdot v_j \geq -1,$$
$$-v_0 \cdot v_i - v_0 \cdot v_j + v_i \cdot v_j \geq -1,$$
$$-v_0 \cdot v_i + v_0 \cdot v_j - v_i \cdot v_j \geq -1,$$
$$v_0 \cdot v_i - v_0 \cdot v_j - v_i \cdot v_j \geq -1,$$

$$\max \frac{1}{4} \sum_{i,j} (3 - v_0 \cdot v_i - v_0 \cdot v_j - v_i \cdot v_j)$$

$$\text{s.t.} \quad v_0 \cdot v_i + v_0 \cdot v_j + v_i \cdot v_j \geq -1, \quad 1 \leq i, j \leq 2n$$

$$v_i \cdot v_{n+i} = -1, \quad 1 \leq i \leq n$$

$$v_i \in \mathbb{R}^{n+1}, \; v_i \cdot v_i = 1, \quad 0 \leq i \leq 2n$$

Рис. 1: SDP-релаксация задачи $\mathsf{MAX2-SAT}$

Алгоритм Гёманса-Уильямсона [2] решает ослабленную версию SDP-релаксации – он не использует "треугольные ограничения". Затем алгоритм *округляет* получившееся решение, используя случайную гиперплоскость. Для округления выбирается случайный вектор $r = (r_0, ..., r_n)$ из стандартного $(n+1)$-мерного нормального распределения. Литералу x_i присваивается значение 0 тогда и только тогда, когда v_i и v_0 лежат по одну сторону от гиперплоскости с вектором нормали r, то есть при $\text{sign}(v_i \cdot r) = \text{sign}(v_0 \cdot r)$.

Алгоритм из работы Фейге и Гёманса [3] решает SDP-релаксацию, приведенную в **рис. 1**. Далее, перед тем, как сделать округление, решения задачи SDP поворачиваются с помощью функции поворота $f : [0, \pi] \to [0, \pi]$, которая удовлетворяет $f(\pi - \theta) = \pi - f(\theta)$ для $0 \leq \theta \leq \pi$. Вектор v_i, образующий угол $\theta_i = \arccos(v_0 \cdot v_i)$ с v_0, преобразуется в вектор v_i', лежащий в плоскости, образованной v_0 и v_i, причем у вектора v_i' угол $\theta_i' = \arccos(v_0 \cdot v_i')$ с v_0, удовлетворяющий $\theta_i' = f(\theta_i)$.

3 Основная часть

3.1 Имплементация алгоритма

Итак, опишем алгоритм который будет имплементирован:

1. Составляем матрицу коэффициентов, стоящих перед $v_i \cdot v_j$.
2. Записываем все ограничения.
3. Решаем задачу с помощью `cvxpy`.
4. Восстанавливаем вектора по матрице из их попарных скалярных произведений через спектральное разложение.
5. Приводим вектора к виду, где $v_0 = (1, 0, ..., 0)$ через QR-разложение.
6. Делаем поворот векторов.
7. Делаем округление случайной гиперплоскостью.

В алгоритме Левина-Ливната-Цвика [1] для поворота используется следующая функция:

$$f(\theta) = 0.588314580\theta + 0.64667394$$

В качестве вектора нормали $r = (r_0, r_1, ..., r_n)$ выбирается такой вектор, что $r_0 = 2$, а r_i являются независимыми стандартными нормальными случайными величинами для $1 \leq i \leq n$.

Код написан на языке программирования Python с использованием библиотек `cvxpy` для решения задачи оптимизации, `numpy` для линейно-алгебраических преобразований, `scipy` для генерации случайных величин. Код алгоритма есть в репозитории на GitHub.

```
import cvxpy as cp
import numpy as np
import scipy
import scipy.stats as sps

def lewin_livnat_zwick(rotation_func, n, clauses_list):
  """
  Имплементация алгоритма Левина-Ливната-Цвика

  :param rotation_func: функция, вычисляющая угол, который будет после
  ↪  поворота
  :param n: число переменных в задаче
  :param clauses_list: список из 2n элементов, обозначающих элементы
  ↪  дизъюнктов
  :return: булевы значения x_0, ..., x_n
  """
  # Вычисляем исходную матрицу весов
  weight_matrix = np.zeros((2 * n + 1, 2 * n + 1))
  for i in range(0, len(clauses_list), 2):
    weight_matrix[clauses_list[i], clauses_list[i + 1]] = 1

  # Вычисляем матрицу весов в задаче
  weight_matrix_modified = weight_matrix.copy()
  weight_matrix_modified_first_column_sum = np.sum(weight_matrix, axis=1)
  weight_matrix_modified_first_row_sum = np.sum(weight_matrix, axis=0)
  weight_matrix_modified[0, :] += weight_matrix_modified_first_column_sum
  weight_matrix_modified[0, :] += weight_matrix_modified_first_row_sum
  C = -weight_matrix_modified / 4

  # Ограничения вида неравенств
  A_inequality = []
  for i in range(1, 2 * n + 1):
    for j in range(1, 2 * n + 1):
      current_A = np.zeros((2 * n + 1, 2 * n + 1))
      current_A[0, i] = 1
      current_A[0, j] = 1
      current_A[i, j] = 1
      A_inequality.append(current_A)
```

```
b_inequality = [-1] * len(A_inequality)

# Ограничения на противоположность знаков v_i и v_{n + i}
A_equality_minus_one = []
for i in range(1, n + 1):
  current_A = np.zeros((2 * n + 1, 2 * n + 1))
  current_A[i, n + i] = 1
  A_equality_minus_one.append(current_A)
b_equality_minus_one = [-1] * len(A_equality_minus_one)

# Ограничения на единичную норму v_i
A_equality_one = []
for i in range(0, 2 * n + 1):
  current_A = np.zeros((2 * n + 1, 2 * n + 1))
  current_A[i, i] = 1
  A_equality_one.append(current_A)
b_equality_one = [1] * len(A_equality_one)

# Переменная задачи
X = cp.Variable((2 * n + 1, 2 * n + 1), symmetric=True)

# Записываем все ограничения
constraints = [X >> 0]
constraints += [
    cp.trace(A_inequality[i].T @ X) >= b_inequality[i] for i in
    ↪  range(len(A_inequality))
]
constraints += [
    cp.trace(A_equality_minus_one[i].T @ X) == b_equality_minus_one[i] for
    ↪  i in range(len(A_equality_minus_one))
]
constraints += [
    cp.trace(A_equality_one[i].T @ X) == b_equality_one[i] for i in
    ↪  range(len(A_equality_one))
]

# Решаем задачу
prob = cp.Problem(cp.Maximize(cp.trace(C.T @ X)), constraints)
prob.solve()

# Получаем вектора по матрице X
eigenvalues, eigenvectors = np.linalg.eigh(X.value)
vectors_transposed = np.sqrt(np.abs(eigenvalues)).reshape(-1, 1) *
↪  eigenvectors.T
vectors = vectors_transposed.T
```

```
    # Получаем через QR-разложение вид, в котором v_0 = (1 0 ... 0)
    Q, R = scipy.linalg.qr(vectors.T)
    if R[0, 0] < 0:
      R = -R
    vectors_transformed = R.T
    vectors_transformed_squeezed = vectors_transformed[:n + 1, :n + 1]

    # Поворачиваем вектора
    vectors_rotated = np.vstack((
        np.array([vectors_transformed_squeezed[0]]),
        np.array([rotate_vector(rotation_func,
         ↪  vectors_transformed_squeezed[0], vectors_transformed_squeezed[i])
         ↪  for i in range(1, n + 1)])
    ))

    # Делаем округления гиперплоскостью
    plane_normal_vector = np.hstack(([2], sps.norm.rvs(size=n)))
    results = [(vectors_rotated[0] @ plane_normal_vector) *
    ↪  (vectors_rotated[i] @ plane_normal_vector) <= 0 for i in range(n + 1)]

    return results
```

Функция `lewin_livnat_zwick(rotation_func, n, clauses_list)` принимает на вход функцию, использующуюся для поворота векторов, число переменных в задаче и набор дизъюнктов. Пример использования можно посмотреть в репозитории на GitHub.

3.2 Тестирование

Код тестов доступен на GitHub. Были проведены тесты для n переменных и дизъюнктов для $n = 5, 10, 15$, в каждом случае использовалось 100 случайно сгенерированных задач. В **табл. 1** приведены статистики для отношений приближения (число выполненных дизъюнктов в результатах алгоритма, деленное на максимальное число дизъюнктов, которые могут быть выполнены). В **табл. 2** приведены статистики для времен работы.

Таблица 1: Результаты отношений приближения для разных n

n	Min	Max	Mean	StdDev	Median	IQR	Outliers
5	0.60	1.00	0.98	0.07	1.00	0.00	8;0
10	0.70	1.00	0.97	0.06	1.00	0.00	21;0
15	0.80	1.00	0.97	0.05	1.00	0.07	11;0

Таблица 2: Результаты времени выполнения для разных n (в секундах)

n	Min	Max	Mean	StdDev	Median	IQR	Outliers
5	0.1174	4.8549	0.2634	0.5941	0.1360	0.0659	0;12
10	0.4449	10.7696	0.7254	1.0200	0.6628	0.2279	0;3
15	1.0244	26.6193	2.3550	4.5296	1.3944	0.2632	0;17

По таблице отношений приближения видно, что полученные результаты всегда высокие и больше 0.940, то есть приближение 0.940 достигается в определенных плохих случаях. В таблице времени выполнения видно ожидаемое его увеличение при увеличении размера задачи.

4 Заключение

Результат Левина-Ливната-Цвика [1] до сих пор не был улучшен. Хостад [6], используя результаты работы Тревисана, Соркина, Судана и Уильямсона [7], показал, что $\forall \varepsilon > 0$, если существует $\left(\frac{21}{22} + \varepsilon\right)$-аппроксимационный алгоритм ($\frac{21}{22} \approx 0.95454$) для $\mathsf{MAX2-SAT}$, то тогда $\mathbf{P} = \mathbf{NP}$.

Список литературы

[1] Lewin, M., Livnat, D., Zwick, U. (2002). Improved Rounding Techniques for the MAX 2-SAT and MAX DI-CUT Problems. In: Cook, W.J., Schulz, A.S. (eds) Integer Programming and Combinatorial Optimization. IPCO 2002. Lecture Notes in Computer Science, vol 2337. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-47867-1_6

[2] Michel X. Goemans and David P. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM 42, 6 (Nov. 1995), 1115–1145. https://doi.org/10.1145/227683.227684

[3] U. Feige and M.X. Goemans. Approximating the value of two prover proof systems, with applications to MAX-2SAT and MAX-DICUT. In Proceedings of the 3rd Israel Symposium on Theory and Computing Systems, Tel Aviv, Israel, pages 182–189, 1995. https://doi.org/10.1109/ISTCS.1995.377033

[4] Matuura, S., Matsui, T. (2001). 0.863-Approximation Algorithm for MAX DICUT. In: Goemans, M., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds) Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques. RANDOM APPROX 2001 2001. Lecture Notes in Computer Science, vol 2129. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44666-4_17

[5] Matuura, S., Matsui, T.: 0.935-approximation randomized algorithm for MAX 2SAT and its derandomization. Technical Report METR 2001-03, Department of Mathematical Engineering and Information Physics, the University of Tokyo, Japan (September 2001)

[6] Johan Håstad. 2001. Some optimal inapproximability results. J. ACM 48, 4 (July 2001), 798–859. https://doi.org/10.1145/502090.502098

[7] L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson. 1996. Gadgets Approximation, and Linear Programming. In Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS '96). IEEE Computer Society, USA, 617.