ЛАБОРАТОРНАЯ РАБОТА №2

Задача о нескольких рюкзаках (Multiple Knapsack Problem)

Дисциплина: Квантовые подходы к обработке информации и искусственному интеллекту

1. ПОСТАНОВКА ЗАДАЧИ

Дано:

- Число предметов: n
- Число рюкзаков: m
- Вес каждого предмета: w_i, где $i = 0, ..., n-1$
- Ценность каждого предмета: v_i, где $i = 0, ..., n-1$
- Вместимость каждого рюкзака: W_k, где $k = 0, ..., m-1$

Требуется найти:

Назначение предметов рюкзакам такое, что:

1. Каждый предмет назначен не более чем одному рюкзаку
2. Для каждого рюкзака k: $\sum_{i \in S_k} w_i \leq W_k$ (не превышаем вместимость)
3. Суммарная ценность взятых предметов максимальна: $\sum_{k=0}^{m-1} \sum_{i \in S_k} v_i$

где S_k — множество предметов в рюкзаке k.

Математическая формулировка:

$$\max_x \sum_{i=0}^{n-1} \sum_{k=0}^{m-1} v_i x_{i,k}$$

при ограничениях:

1. Каждый предмет в не более чем одном рюкзаке: $\sum_{k=0}^{m-1} x_{i,k} \leq 1, \quad \forall i$
2. Вместимость каждого рюкзака: $\sum_{i=0}^{n-1} w_i x_{i,k} \leq W_k, \quad \forall k$

где $x_{i,k} = 1$ означает, что предмет i в рюкзаке k.

Сложность: Задача является NP-трудной и сложнее обычной задачи о рюкзаке, так как добавляется измерение выбора рюкзака. Динамическое программирование имеет сложность $O(n \prod_{k=0}^{m-1} W_k)$, что быстро становится неприемлемым.

2. QUBO-ФОРМУЛИРОВКА

2.1. Бинарные переменные

$$x_{i,k} \in \{0,1\}, \quad i = 0, ..., n-1,\ k = 0, ..., m-1$$

Интерпретация: $x_{i,k} = 1$ означает, что предмет i помещён в рюкзак k.

Число переменных: $n \cdot m$

Дополнительная переменная для "не брать":
Можно добавить фиктивный рюкзак $k = m$ (рюкзак "мусор") для предметов, которые не берём, но проще контролировать это ограничением $\sum_k x_{i,k} \leq 1$.

2.2. Преобразование в задачу минимизации

Максимизация $\sum_{i,k} v_i x_{i,k}$ эквивалентна минимизации $-\sum_{i,k} v_i x_{i,k}$:

$$E_{obj} = -\lambda \sum_{i=0}^{n-1} \sum_{k=0}^{m-1} v_i x_{i,k}$$

где $\lambda > 0$ — коэффициент масштабирования (обычно $\lambda = 1$).

2.3. Ограничения

Ограничение 1: Каждый предмет в не более чем одном рюкзаке

$$\sum_{k=0}^{m-1} x_{i,k} \leq 1, \quad \forall i$$

Штраф за нарушение:

$$E_1 = A \sum_{i=0}^{n-1} \left(\sum_{k=0}^{m-1} x_{i,k} \right) \left(\sum_{k=0}^{m-1} x_{i,k} - 1 \right)$$

Альтернативно:

$$E_1 = A \sum_{i=0}^{n-1} \sum_{k<k'} x_{i,k} x_{i,k'}$$

Ограничение 2: Вместимость каждого рюкзака

$$\sum_{i=0}^{n-1} w_i x_{i,k} \leq W_k, \quad \forall k$$

Штраф за превышение:

$$E_2 = B \sum_{k=0}^{m-1} \left(\sum_{i=0}^{n-1} w_i x_{i,k} - W'_k \right)^2$$

где $W'_k = \alpha \cdot W_k$ с $\alpha \in [0.85, 0.95]$ — эмпирический совет.

Раскрывая квадрат для каждого рюкзака:

$$E_2 = B \sum_{k=0}^{m-1} \left[\sum_{i,j=0}^{n-1} w_i w_j x_{i,k} x_{j,k} - 2W'_k \sum_{i=0}^{n-1} w_i x_{i,k} + (W'_k)^2 \right]$$

Константу $(W'_k)^2$ можно опустить.

2.4. Итоговая энергия

$$E_{total}(x) = E_{obj}(x) + E_1(x) + E_2(x)$$

$$E_{total}(x) = -\lambda \sum_{i=0}^{n-1} \sum_{k=0}^{m-1} v_i x_{i,k} + A \sum_{i=0}^{n-1} \sum_{k<k'} x_{i,k} x_{i,k'} + B \sum_{k=0}^{m-1} \left[\sum_{i,j=0}^{n-1} w_i w_j x_{i,k} x_{j,k} - 2W'_k \sum_{i=0}^{n-1} w_i x_{i,k} \right]$$

Задача: $\min_{x \in \{0,1\}^{n \times m}} E_{total}(x)$

3. МЕТОДЫ РЕШЕНИЯ

3.1. Динамическое программирование (DP)

Идея: Обобщение классического DP для одного рюкзака на несколько рюкзаков.

Подход 1 (полное DP):

- Таблица $dp[i][(w_0, w_1, ..., w_{m-1})]$ — максимальная ценность для первых i предметов и остаточных вместимостей рюкзаков
- Сложность: $O(n \prod_{k=0}^{m-1} W_k)$ — быстро становится неприемлемой

Подход 2 (последовательное DP):

1. Решить задачу для первого рюкзака, получить S_0
2. Исключить S_0 из предметов
3. Решить для второго рюкзака из оставшихся, получить S_1
4. И так далее

Сложность: $O(m \cdot n \cdot W_{max})$, где $W_{max} = \max_k W_k$

Недостаток: Последовательный подход не гарантирует глобального оптимума.

Применимость: Эффективен только для малых задач ($m \leq 3$, $W_k \leq 1000$).

3.2. Имитация отжига (Simulated Annealing)

Идея: Метод глобальной стохастической оптимизации, основанный на аналогии с физическим процессом отжига металлов. Алгоритм начинает с высокой "температуры", при которой допускаются переходы в состояния с большей энергией (для исследования пространства решений и избежания локальных минимумов). Затем температура постепенно снижается, и алгоритм становится более "жадным".

Основные компоненты:

1. **Температурный график:** Управляет вероятностью принятия ухудшающих решений

   $$T(t) = T_0 \cdot \alpha^t$$

   где T_0 — начальная температура, $\alpha \in (0, 1)$ — коэффициент охлаждения, t — номер итерации
2. **Критерий принятия решения:** Для перехода из состояния с энергией E в состояние с энергией E':
   - Если $\Delta E = E' - E < 0$ (улучшение) — принять всегда
   - Если $\Delta E \geq 0$ (ухудшение) — принять с вероятностью:

   $$P(\text{accept}) = e^{-\Delta E / T}$$

   При высокой температуре T эта вероятность близка к 1 (принимаем почти любые ухудшения). При низкой температуре $T \to 0$ эта вероятность стремится к 0 (становимся жадными).
3. **Генерация соседних состояний:** Зависит от задачи (перестановки, перевороты битов, изменения цветов и т.д.)

Параметры для подбора:

- **Начальная температура** T_0: Обычно подбирается так, чтобы в начале принималось около 80% ухудшающих переходов. Типичный диапазон зависит от масштаба энергии задачи.
- **Коэффициент охлаждения** $\alpha \in [0.93, 0.99]$: Чем ближе к 1, тем медленнее охлаждение и тем дольше работает алгоритм. Типичные значения: 0.95-0.97.
- **Критерий остановки:** Обычно $T < T_{min}$ (например, $T_{min} = 0.001$) или фиксированное число итераций.

3.3. Поиск в больших окрестностях (Large Neighborhood Search)

Идея: Фиксировать большую часть назначений и переоптимизировать блок предметов.

Основные компоненты:

- Блок размера $p \approx n/4$ предметов
- Фаза разрушения: извлечь p случайных предметов из их рюкзаков
- Фаза восстановления: переназначить эти p предметов оптимально (SA или жадно)
- Критерий принятия: улучшение целевой функции

Алгоритм:

1. Получить начальное назначение (жадное)
2. Основной цикл:
    - Выбрать случайный блок из p предметов
    - "Открепить" их (обнулить все $x_{i,k}$ для i из блока)
    - Оптимизировать назначение только этих p предметов (с помощью SA)
    - Если улучшение — принять
3. Вернуть лучшее решение

3.4. Жадный алгоритм (Baseline)

Идея: Сортировать предметы по эффективности и последовательно назначать рюкзакам.

Алгоритм:

1. Вычислить для каждого предмета: $r_i = v_i/w_i$
2. Отсортировать предметы по убыванию r_i
3. Для каждого предмета по порядку:
    - Попробовать поместить в рюкзак с максимальной остаточной вместимостью
    - Если не помещается ни в один — пропустить
4. Вернуть назначение

Свойства: Быстрый ($O(n \log n + nm)$), но не гарантирует оптимальности.

3.5. Жадный с локальными улучшениями

Улучшение жадного:

1. Получить жадное решение
2. Локальные улучшения:
   - Попытки переместить предметы между рюкзаками
   - Попытки обменять предметы между рюкзаками
   - Попытки добавить непомещённые предметы
3. Применить улучшение при увеличении ценности с сохранением допустимости
4. Повторять до отсутствия улучшений

4. МЕТРИКИ СРАВНЕНИЯ

4.1. Основные метрики

Суммарная ценность (целевая метрика):

$$V_{total} = \sum_{i=0}^{n-1} \sum_{k=0}^{m-1} v_i x_{i,k}$$

Вес каждого рюкзака:

$$W_{used,k} = \sum_{i=0}^{n-1} w_i x_{i,k}, \quad k = 0, ..., m-1$$

Коэффициент заполнения каждого рюкзака:

$$\text{Utilization}_k = \frac{W_{used,k}}{W_k} \times 100\%$$

Средняя утилизация:

$$\text{Avg Utilization} = \frac{1}{m} \sum_{k=0}^{m-1} \text{Utilization}_k$$

Баланс заполнения:

$$\text{Balance} = \frac{\sigma(\text{Utilization}_0, ..., \text{Utilization}_{m-1})}{\text{Avg Utilization}}$$

где σ — стандартное отклонение. Меньше = лучше баланс.

4.2. Отличие от оптимума

Отличие от динамического программирования (если применимо):

$$\text{Gap}_{DP}(\%) = \frac{V_{DP} - V_{method}}{V_{DP}} \times 100\%$$

Отличие от жадного алгоритма:

$$\text{Gap}_{greedy}(\%) = \frac{V_{greedy} - V_{method}}{V_{greedy}} \times 100\%$$

(отрицательный gap означает улучшение)

4.3. Сравнение времени выполнения

Ключевое сравнение: Время QUBO-методов vs время динамического программирования (если применимо).

Измерять для каждого метода:

- Время выполнения (среднее по 10 запускам)
- Время до первого допустимого решения
- Время до лучшего найденного решения

Анализ: При каких параметрах (n, m, W_k) DP становится неприменимым?

5. ПОДБОР ПАРАМЕТРОВ

5.2. Добавление собственных модификаций

Возможные улучшения:

1. **Штраф за недогруз рюкзаков:**
   Поощрение высокой утилизации:

$$E_{underuse} = C \sum_{k=0}^{m-1} (W_k - \sum_i w_i x_{i,k})^2$$

2. **Штраф за дисбаланс заполнения:**

$$E_{balance} = D \cdot \mathrm{Var}(\mathrm{Utilization}_0, ..., \mathrm{Utilization}_{m-1})$$

Требование: Обоснование и демонстрация улучшения результатов.

6. ПЛАН ЭКСПЕРИМЕНТОВ

6.1. Требования

1. Запустить каждый метод **минимум 10 раз**
2. Зафиксировать random seed
3. Для каждого запуска записать:
    - Суммарную ценность
    - Вес каждого рюкзака
    - Допустимость решения
    - Утилизацию каждого рюкзака
    - Баланс заполнения
    - Время выполнения

6.2. Обязательные эксперименты

Эксперимент 1: Сравнение методов

- SA vs LNS vs Greedy vs DP (если применимо)
- Метрики: ценность, время, допустимость, утилизация

Эксперимент 2: Влияние параметров A и B

- Зафиксировать всё кроме A и B
- Варьировать $A \in [10, 30, 50, 100]$, $B \in [5, 10, 20, 40]$
- Построить тепловую карту: допустимость vs (A, B) и ценность vs (A, B)

Эксперимент 3: Влияние параметра α (смягчение)

- Зафиксировать A, B
- Варьировать $\alpha \in [0.8, 0.85, 0.9, 0.95, 1.0]$
- Проанализировать влияние на утилизацию и допустимость

Эксперимент 4: Масштабируемость (если возможно)

- Сравнить время: DP vs SA vs LNS
- Для разных размеров задачи (если DP применим)

6.3. Анализ результатов

Обязательные элементы:

1. **Таблица результатов:**
   - Метод | Ценность (min/avg/max/std) | Время | Допустимость се
2. **Статистический анализ:**
   - Какой метод лучше по ценности?
   - Достигается ли высокая утилизация?
   - Достигается ли баланс между рюкзаками?
   - Соотношение качество/время

7. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чём отличие задачи о нескольких рюкзаках от классической задачи о одном рюкзаке? Почему она сложнее?
2. Почему динамическое программирование становится неэффективным при увеличении числа рюкзаков m?
3. Объясните, как квадратичный штраф E_1 гарантирует, что каждый предмет помещён не более чем в один рюкзак.
4. Как штраф E_2 кодирует ограничение вместимости для каждого рюкзака?
5. Почему используется смягчённое ограничение $W'_k = \alpha W_k$ с $\alpha < 1$? Какие преимущества и недостатки?
6. Что произойдёт, если параметр A слишком маленький? А если B слишком маленький? Покажите на ваших экспериментах.
7. В чём преимущество имитации отжига перед жадным алгоритмом для Multiple Knapsack?
8. Как поиск в больших окрестностях помогает улучшить решение? Опишите механизм разрушения и восстановления.
9. Опишите процесс подбора параметров A, B, α в вашей реализации. Какие значения вы выбрали и почему?
10. Достигается ли в вашем лучшем решении высокая утилизация всех рюкзаков? Достигается ли баланс заполнения?
11. Сравните время выполнения DP (если применимо) и SA на вашем варианте. При каких параметрах DP становится неприменимым?

12. Как можно модифицировать задачу, если разные рюкзаки имеют разную "стоимость использования"?

УДАЧИ В РАБОТЕ!