

Лекция 14

Эвристические алгоритмы

План лекции

- Эвристические алгоритмы
- Жадные эвристики для задачи коммивояжёра
- Локальный поиск
 - Общий принцип
 - Варианты лок.поиска для ЗК
 - Алгоритм Метрополиса и имитация отжига

Эвристические алгоритмы

Эвристический алгоритм — это алгоритм решения задачи, правильность которого для всех возможных случаев не доказана, но про который известно, что он даёт достаточно хорошее решение в большинстве случаев. (Википедия)

Иногда термин «эвристика» используется также для обозначения методов ускорения заведомо точных методов (например, полного перебора).

Эвристические алгоритмы

Наиболее частые эвристики:

- Жадный алгоритм
- Ограниченный перебор только перспективных вариантов
- Последовательное улучшение («локальный поиск»)

В т.ч. в сочетании с вероятностным выбором.

Жадные алгоритмы

Жадный алгоритм — однопроходный итерационный алгоритм. Строит решение, добавляя на каждом шаге к текущему *частичному* решению новый элемент. Добавляемый элемент выбирается на основе локального оптимума («наилучший на текущем шаге»).

Жадные алгоритмы:

- Дают точное решение для задач на матроидах (с аддитивной целевой функцией)
- Для некоторых задач дают приближённое решение с гарантированной (не обязательно константной) оценкой приближения
- В общем случае используются как эвристики

Рассмотрим жадные эвристики для «Задачи коммивояжёра».

Жадные алгоритмы для ЗК

Метод ближайшего соседа

Последовательно строим маршрут, переходя из каждой вершины в ближайшую к ней новую вершину.

1. Выберем стартовую вершину (случайно или фиксированно).
2. Из текущей вершины переходим в ближайшую к ней ещё не посещённую вершину.
3. Если прошли по всем вершинам, то возвращаемся в стартовую, иначе — goto 2.

Временная сложность: $O(n^2)$

Обычный вес полученного решения: на 25% выше оптимального (т. е. $R = 1.25$)

Жадные алгоритмы для ЗК

Жадный выбор рёбер

Добавляем в цикл самое дешёвое из допустимых рёбер. Ребро допустимо, если при его добавлении к маршруту

- Не образуется цикл меньше чем из n вершин.
- Степень вершины в маршруте не становится > 2 .

1. Упорядочить рёбра по неубыванию весов.
2. Последовательно по порядку (п.1) перебираем рёбра.
3. Если ребро допустимо — добавляем его в цикл, иначе - пропускаем.

Временная сложность: $O(n^2 \log n)$

Обычный вес полученного решения: на 15-20% выше оптимального.

Жадные алгоритмы для ЗК

Жадные вставки

Последовательно вставляем в частичное решение (путь, цикл) новые вершины.

1. Построить начальный путь (ребро минимального веса) или цикл («треугольник» минимального веса).
2. Пока в решение входят не все вершины:
 1. Выбрать новую вершину — ближайшую к какой-либо вершине цикла/пути
 2. Выбрать пару смежных (в цикле/пути) вершин, между которыми лучше всего вставить новую вершину.
 3. Вставить новую вершину

Временная сложность: $O(n^2)$

Обычный вес полученного решения: на 25% выше оптимального.

Локальный поиск

Локальный поиск — итерационный метод (*метаэвристика*), при котором выбирается начальное решение и постепенно улучшается до тех пор, пока улучшения возможны. Варианты улучшения строятся с помощью семейства преобразований $F = \{f: S(x) \rightarrow S(x)\}$.

Множество $N(s) = \{s' = f(s): f \in F\}$ называется *окрестностью* допустимого решения s .

Идея локального поиска: строим начальное решение и последовательно его улучшаем, выбирая новое решение из окрестности текущего. Когда улучшение невозможно — останавливаемся. Текущее решение — *локальный оптимум*.

Достоинства:

- Универсальность. Общая схема, легко обобщающаяся на различные, в т.ч. новые, оптимизационные задачи.
- Алгоритм с *отсечением по времени* (*anytime algorithm*). Быстро находит допустимое решение, а потом использует всё имеющееся время для его улучшения.

Локальный поиск

Общая схема (для задачи минимизации)

1. Выбрать $s \in S(x)$.
2. Среди $\{s' = f(s): f \in F\}$ найти $s': s' \in S(x)$ и $w(s') < w(s)$
 - Если найдено, то $s := s'$; перейти к п. 2
 - Иначе — остановиться и вернуть текущее s как решение

Для п.2 есть два варианта реализации:

- (a) выбрать первое подходящее s'
- (b) выбрать наилучшее s' (с наименьшим весом).

Исследования показывают, что обычно варианты (a) и (b) примерно равны по качеству получаемого решения, при этом (a) работает быстрее.

На скорость и качество решения влияет также размер окрестности (чем меньше — тем выше скорость, но ниже качество).

Локальный поиск для ЗК

Рассмотрим симметричную ЗК. Допустимые решения: $S(x)$ = множество гамильтоновых циклов на заданном графе.

Транспозиции

В качестве преобразований $F = \{f\}$ рассматриваются транспозиции элементов в последовательности вершин гамильтонова цикла.

Например, для $s = (1, 2, 3, 4, 5, 6, 7)$ окрестность будет содержать циклы: $(1, 2, \underline{4}, \underline{3}, 5, 6, 7)$, $(\underline{5}, 2, 3, 4, \underline{1}, 6, 7)$ и $(1, \underline{7}, 3, 4, 5, 6, \underline{2})$.

Всего $n(n-1)/2 = O(n^2)$ соседей.

Локальный поиск для ЗК

Пусть w_{max} и w_{ave} — соответственно максимальный и средний вес гамильтонова цикла.

Пусть K — константа, такая что $w_{max} < K w_{ave}$.

Теорема (Gover, 1992). Локальный поиск на основе транспозиций, начиная со случайного гамильтонова цикла, достигает локального оптимума за время $O(nK)$



Аналогичное утверждение справедливо для локального поиска для задачи о разбиении. Окрестность допустимого решения состоит из всех вариантов, получаемых инверсией (переносом в другое подмножество) любого элемента.

Локальный поиск для ЗК

Эвристика k-opt (k-обмен)

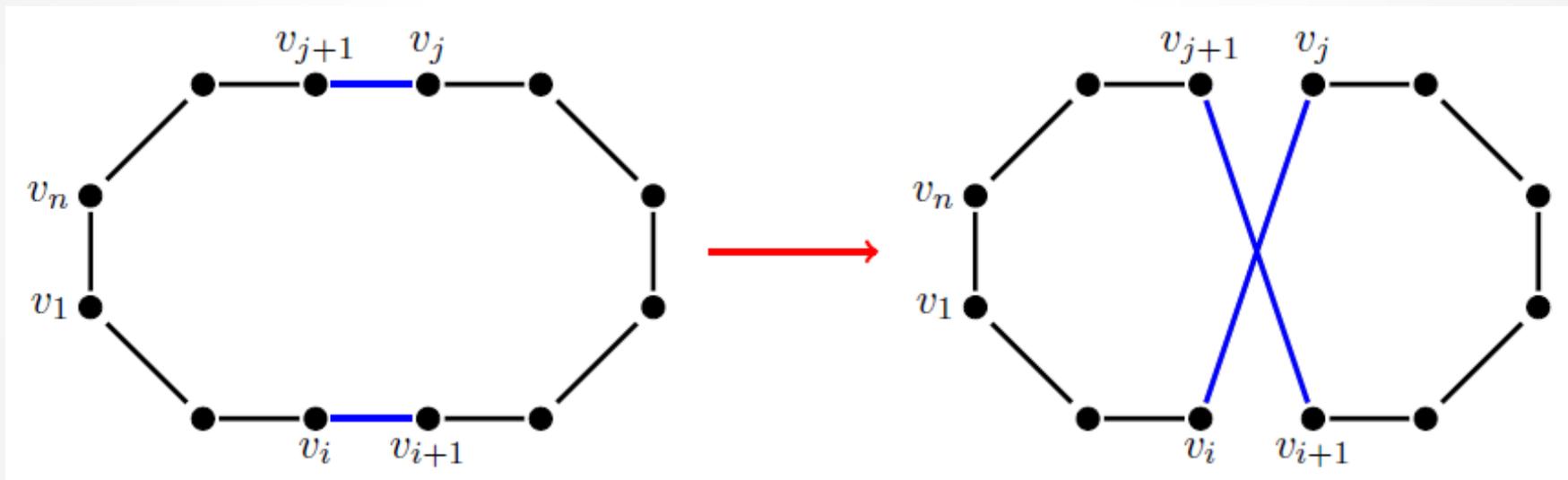
В качестве преобразований $F = \{f\}$ рассматриваются преобразования такого вида:

- 1) В текущем цикле выбрать k несмежных рёбер
- 2) Удалить выбранные рёбра из цикла
- 3) Концы рёбер соединить попарно другим способом.

Чаще всего используется 2-opt или 3-opt.

Локальный поиск для ЗК

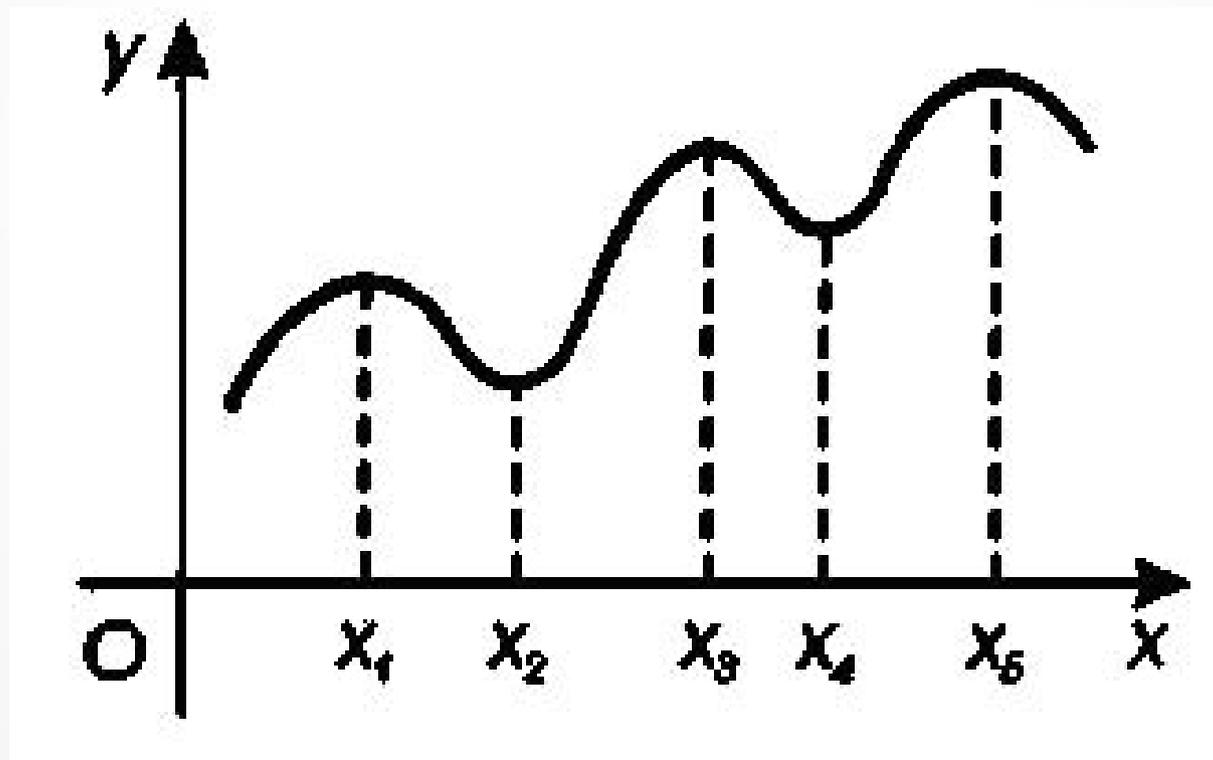
Пример преобразования 2-opt



Как удобнее реализовать такое преобразование в программе?

Эвристические алгоритмы

Недостатки «жадных» алгоритмов и локального поиска: они могут «скатиться» в локальный оптимум.



Эвристические алгоритмы

Выход - разрешить алгоритму переходить к решению с худшим весом, чем текущее решение.

- Детерминированный выбор — выбираем наилучшего соседа, даже если он хуже, чем текущий вариант.
- Добавляем в процедуру случайные шаги.

Эвристические алгоритмы

При детерминированном выборе логичным будет в качестве следующего значения s выбирать наилучшее по весу s' из $N(s)$. Даже если $w(s') > w(s)$ (для задачи минимизации).

Проблема: такой алгоритм может зацикливаться.

Решение: *поиск с запретами* (tabu search).

Эвристические алгоритмы

Поиск с запретами

Исключить (или уменьшить вероятность) зацикливания, запретив

- повторно переходить к s , в котором недавно были, либо
- повторно выполнять преобразования f , которые недавно выполнялись

Для реализации нужен список запретов (tabu list), содержащий k последних решений или преобразований. Размер списка (k) обычно определяется эмпирически.

Эвристические алгоритмы

TabuSearch:

$T := []$

$s := \text{InitialSolution}(x)$

while (*условие продолжения поиска*):

$s' := \text{наилучший из } N(s) \setminus T$

 Добавить s' в T

$s := s'$

Эвристические алгоритмы

Случайный поиск

Варианты случайного локального поиска:

- Случайный выбор начального варианта
- Случайный (рандомизированный) выбор элемента из окрестности

Эвристические алгоритмы

Случайный выбор начального варианта

Начальный вариант (который будет потом итерационно улучшаться) выбираем с помощью случайного алгоритма.

RandomizedLocalSearch:

```
Record = NULL
```

```
for i=1 to IterCount:
```

```
    s := RandomSolution(x)
```

```
    s' := LocalSearch(s)
```

```
    if w(s') < w(Record)
```

```
        Record := s'
```

Эвристические алгоритмы

Случайный выбор элемента из окрестности

Для каждого $s' \in N(s)$ вероятность выбора зависит от $w(s')$.

Т.е. лучшие варианты выбираются из $N(s)$ с большей вероятностью, но и (локально) менее предпочтительные (но способные привести к глобальному оптимуму) варианты также могут быть выбраны с ненулевой вероятностью.

Алгоритм Метрополиса

Один из популярных вариантов — алгоритм **Метрополиса**.

Идея: разрешаем алгоритму ухудшать текущее состояние, но с вероятностью, зависящей от качества решения.

Метафора: переход физической стохастической системы из одного состояния в другое. Такая система «стремится» перейти в состояние минимальной потенциальной энергии.

Алгоритм Метрополиса

Вероятность перехода из состояния s в состояние s' :

$$e^{-\Delta w / (kT)}, \text{ где } \Delta w = w(s') - w(s).$$

Параметры алгоритма: константы k и T («температура»).

Один шаг алгоритма Метрополиса:

1. Пусть s — текущее состояние. Случайно выбрать $s' \in N(s)$.
2. Если $w(s') < w(s)$, то $s := s'$.
3. Иначе: обновить s ($s := s'$) с вероятностью $e^{-\Delta w / (kT)}$
4. При необходимости, обновить **рекорд**.

Алгоритм Метрополиса

Теорема.

Пусть $f_s(t)$ -доля первых t шагов, в течение которых система пребывает в состоянии s . Тогда предел $f_s(t)$ при $t \rightarrow \infty$ с вероятностью 1 равен

$$\frac{1}{Z} \cdot e^{-w(s)/(kT)}$$

Здесь $Z = \sum_{s \in S(x)} e^{-w(s)/(kT)}$.

Т.е. алгоритм «стремится» чаще переходить в состояния с меньшим весом («энергией»).

Алгоритм Метрополиса

Проблема алгоритма Метрополиса: в состояниях, близких к оптимальному (текущее покрытие содержит мало вершин), большинство соседних состояний $s' \in N(s)$ ухудшают вес решения. Но алгоритм будет переходить в такие состояния с достаточно большой вероятностью (потому что их много).

Решение: по мере приближения к оптимуму (=с течением времени) снижать вероятность перехода в ухудшающие состояния. Эта вероятность определяется «температурой» - параметром T .

Имитация отжига

Алгоритм имитации отжига (simulated annealing)

1. Установим начальное значение T .
2. Выполняем алгоритм Метрополиса, на каждом шаге уменьшая температуру T в соответствии с *планом охлаждения*. Например: $T := \alpha T$, где α - заданный коэффициент (гиперпараметр), $\alpha < 1$.

Выполнение завершается после заданного количества итераций.