Тема - Задача коммивояжёра и её методы решения

Оптимальное управление

Докладчик:
Сюй Минчуань
Совместный Университет МГУ-ППИ в Шэньчжэне
Факультет ВМК, III курс

13 июля 2020 г.

Содержания доклада

Имеется N городов, занумерованных числами $1, 2, \ldots, N$. Для любой пары городов (i, j) задано расстояние c_{ij} (критерий выгодности: расстояние, время, стоимость, совокупный критерий и.т.д.) между ними. Начиная с какого-то города, коммивояжер должен побывать во всех городах **ровно по одному разу**, и вернуться в начальный город. Требуется найти минимальный маршрут.
Будем считать, что $c_{ii} = +\infty$, $i = 1, 2, \ldots, n$. Таким образом, зачада коммивояжера (ЗКВ) сводится к задачи минимизации

$$J(u) = c_{1 i_1} + c_{i_1 i_2} + \ldots + c_{i_{N-1} 1} \to min \qquad (1)$$

на множестве допустимых маршрутов $u = (1, i_1, i_2, \ldots, i_{N-1}, 1)$, где $(i_1, i_2, \ldots, i_{N-1})$ - любая перестановка чисел $2, 3, \ldots, N$. Удобнее записать ЗКВ в матрице смежности:

$$C = \{c_{ij}\} = \begin{pmatrix} \infty & c_{12} & c_{13} & \cdots & c_{1N} \\ c_{21} & \infty & c_{23} & \cdots & c_{2N} \\ c_{31} & c_{32} & \infty & \cdots & c_{3N} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{N1} & c_{N2} & c_{N3} & \cdots & \infty \end{pmatrix}$$

Дискретная задача оптимального управления (вспомогательная)

$$\begin{cases} J\left(x,[u_i]_k\right) = \sum_{i=k}^{N-1} G_i\left(x_i,u_i\right) + \Phi\left(x_N\right) \to \text{ inf} \\ x_{i+1} = F_i\left(x_i,u_i\right), i = k, k+1, \ldots, N-1, \quad x_k = x \\ x_i \in X_i, \quad i = k, k+1, \ldots, N \\ [u_i]_k = \left(u_k, u_{k+1}, \ldots, u_{N-1}\right), u_i \in V_i, \quad i = k, k+1, \ldots, N-1 \end{cases} \tag{2}$$

где $k \in \{0, 1, 2, \ldots, N-1\}$, и $x \in X_k$.

Через $\Delta_k(x)$ обозначим множество всех управлений $[u_i]_k$ таких, что:

- $u_i \in V_i, \quad i = k, k+1, \ldots, N-1.$
- $[x_i]_k = (x_k = x, x_{k+1}, \ldots, x_N)$, где $[x_i]_k$ - соответствующие траектории, удовлетворяющие фазовым ограничениям: $x_i \in X_i, \quad i = k, k+1, \ldots, N$

Функция Беллмана дискретной задачи

$$B_k(x) = \inf_{[u_i]_k \in \Delta_k(x)} J_k\left(x,[u_i]_k\right), \quad k = 0, 1, \ldots, N-1 \tag{3}$$

Областью определения этой функции является $\mathcal{X}_k = \{x \in X_k : \Delta_k(x) \neq \varnothing\}$.

Функция Беллмана дискретной задачи удовлетворяет **уравнению Беллмана**.

Принцип оптимальности

Принцип оптимальности в общем случае

Если $[u_i]_* = \left(u_0^*, \ldots, u_{N-1}^*\right), [x_i]_* = (x_0^*, \ldots, x_N^*)$ - оптимальные управление и траектория исходной задачи, то в любой вспомогательной задаче при $x_k = x_k^*$ оптимальными будут управление $[u_i]_k = \left(u_k^*, \ldots, u_{N-1}^*\right)$ и траектория $[x_i]_k = \left(x_k^*, \ldots, x_N^*\right)$.

Принцип оптимальности для ЗКВ

Обозначим через $B_{k-1}(1, i_1, \ldots, i_k)$ функцию Беллмана, равную длине самого короткого из маршрутов, соединяющих города 1 и i_k и проходящих в любом порядке через города $i_1, \ldots, i_k$. Принцип оптимальности, примененный в ЗКВ, дает такой результат: функция Беллмана при всех $k = 0, 1, \ldots, N-1$ удовлетворяет уравнению Беллмана

$$B_k(1, i_1, \ldots, i_k, i_{k+1}) = \min \left\{ \begin{array}{c} B_{k-1}(1, i_2, i_3, i_4, \ldots, i_{k-1}, i_k, i_1) + c_{i_1 i_{k+1}} \\ B_{k-1}(1, i_1, i_3, i_4, \ldots, i_{k-1}, i_k, i_2) + c_{i_2 i_{k+1}} \\ B_{k-1}(1, i_1, i_2, i_4, \ldots, i_{k-1}, i_k, i_3) + c_{i_3 i_{k+1}} \\ \ldots \\ B_{k-1}(1, i_1, i_2, i_3, \ldots, i_{k-2}, i_k, i_{k-1}) + c_{i_{k-1} i_{k+1}} \\ B_{k-1}(1, i_1, i_2, i_3, \ldots, i_{k-2}, i_{k-1}, i_k) + c_{i_k i_{k+1}} \end{array} \right\} \quad (4)$$

Вычислительная схема

То есть, зная оптимальное решение задачи коммивояжера для **k городов**, мы можем очень легко найти решение задачи коммивояжера, получающейся добавлением к ней **(k+1)-го города**: оптимальное движение по $k+1$ городу повторит оптимальное движение по k городам.

Иными словами, мы можем последовательно решить подзадачу, при этом не теряем оптимальность решения. Очевидно, этот метод требует больше памяти при расчёте, но разложив исходную задачу в подзадачи, мы значительно уменьшаем время для получения решения по сравнению с методом перебора, который таже с помошью суперкомпьютера в небольшом размерности задачи нельзя решить задачу в разумное время.

Задача коммивояжера является **NP-полной** проблемой, к которой можно свести любую другую задачу из класса NP за полиномиальное время. Таким образом, NP-полные задачи образуют в некотором смысле подмножество «самых сложных» задач в классе NP; и если для какой-то из них будет найден быстрый алгоритм решения, то и любая другая задача из класса NP может быть решена так же быстро.

Практическая реализация

Основная часть реализации динамического программирования

```
int d[n][1<<(n-1)]; //1<<(n-1) - 2^(n-1) Process matrix
for(int i=1;i<n;i++) // initialize distance from every city to city0
    d[i][0]=dis[i][0];

for(int j=1;j<1<<(n-1);j++){
    for(int i=0;i<n;i++){ //find the shortest distance from city i by passing cities set j
        if(((1<<(i-1))&j)==0){ //if city i is not in set j
            minDis=50000;
            for(int k=1;k<n;k++){
                if(((1<<(k-1))&j)!=0) //if city k is in set j
                    //test every possible route from i to the next city k
                    //j-(1<<(k-1)) - kick out from set j
                    minDis = min(minDis,dis[i][k]+d[k][j-(1<<(k-1))]);
            }
            d[i][j] = minDis;
        }
    }
}
```

Содержания доклада

Классы методов для ЗКВ

Конечно, для этой классической задачи разработаны многие достадочно эффективные методы. В общем можем их разделять на два класса: **точные методы** и **приближённые**. К точным методам известны как

- Динамическое программирование
- Метод ветвей и границ
- Жадный алгоритм

К приближённым методам относится ряд **эвристичеких методов** как

- Генетический алгоритм
- Алгоритм имитации отжига
- Муравьиный алгоритм
- Нейронные сети

и.т.д.

Генетический алгоритм

Сейчас рассмотрим более интересные методы. **Генетический алгоритм** основан на идее эволюции с помощью **естественного отбора**.

установливается начальное поколение $k = 0$
вероятность проведения кроссовера = α
вероятность мутации = β
построить популяцию из n случайно сгенерированных особей P_k
while не остановится **do**
    оценивание: вычислять приспособленность(i) для каждой особи P_k
    выбор: выбирать m членов P_k и вставлять в P_{k+1}
    кроссовер: генерировать αm потомков кроссовером и вставлять в P_{k+1}
    мутация: генерировать βm потомков мутацией и вставлять в P_{k+1}
    обновление поколения: $k = k + 1$
end while
вернуть сильнейшую особь из P_{last}

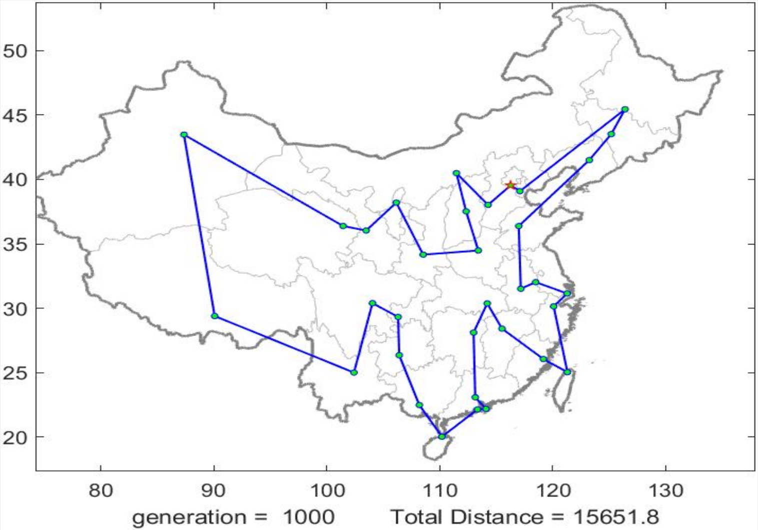
В отличие от градиентного метода, генетические алгоритмы не «застревают» в точках локального экстремума, а позволяют найти «глобальный» минимум.

Алгоритм имитации отжига

Алгоритм вдохновлён **процессом отжига** в металлургии — техники, заключающейся в нагревании и контролируемом охлаждении металла, чтобы увеличить его кристаллизованность и уменьшить дефекты.

установливается начальное приближение x_0, и $x^{\text{current}} = x_0$
установливается начальное температура $T = T_0$
while не остановится do
  for $i = 1$ to T_L do
    случайно генерировать соседное решение $x' \in N(x^{\text{current}})$
    вычислять изменение cost $\Delta C = C(x') - C(x^{\text{current}})$
    if $\Delta C \leq 0$ or random $(0, 1) < \exp\left(-\frac{\Delta C}{kT}\right)$ then
      $x^{\text{current}} = x'$ [принять новое состояние]
    end if
  end for
    установливается новая температура $T = \text{decrease}(T)$ [понижать температуру]
end while
вернуть решение, соответствующее минимуму функции cost

Пример: Кратчайший путь экскурсии по Китаю



Спасибо за внимания!
Вопросы?