第十二讲 在线算法

主讲教师: XXX
讲义整理: XXX

2021 年 1 月 10 日

1 Competitive Analysis

在线算法的性能通过一种类似于近似比的概念来衡量，即 **competitive ratio**;

若 A 表示在线算法，OPT 表示相同问题的最优离线 (offline) 算法，
对于输入 I，A(I) 表示该算法的消耗，OPT(I) 表示最优解的消耗，则 competitive ratio 定义如下：
设 c 为 competitive ratio，对于最小化问题，$\forall I$，存在常数 β 使得

$$A(I) \leq c * OPT(I) + \beta$$

同时，我们称这一算法为 c-competitive 算法。

2 k-Server Problem

2.1 特殊情况:paging problem

首先考虑一个操作系统或体系结构中的常见问题，它是 k-Server Problem 的一种特殊情况；

有容量为 k pages 的 cache 以及容量为 n pages 的主存；

访问 cache 中的数据没有消耗，而如果访问的数据不在 cache 中，则必须将其从主存中移动到 cache 中，记消耗为 1。

目标：对于一段请求序列，要求总 cost 最小化。

2.2 问题描述

对于一个加权网络 $G = (V, E)$，有 K 个 server 在初始分布在某些顶点上;
请求集合 $\sigma = (r_1, r_2, ..., r_n)$ 逐次到来（每个请求发生在一个顶点上）

当一个请求发生时，必须有一个 server 存在于这个顶点上，否则就需要分配一个 servver 前往这个顶点；
目标：将总的路程消耗（每个 server 移动路程的总和）最小化。

可以注意到，当网络为完全图时，且所有边权均为 1 时，k-Server Problem 即:paging problem。

2.3 贪心算法

k-Server Problem 的贪心算法：当每次请求发生时，派遣离这一顶点最近的 server 前往该顶点。

这一贪心算法并不能提供一个可观的解，考虑如下最坏情况：

对于一个只有三个顶点的图，同时有两个 servers(server 可以位于任意位置)，顶点 C 位于顶点 A，B 之间，且 $d(A,C) < d(B,C)$

发生的请求序列为：$\sigma = (A, B, C, A, C, A, C, A, C$......)

可以注意到，这种情况下会存在一个 servers 不断往返于 A,C 之间，而位于 B 的 server 并不会移动，而最优解显然是两个 servers 位于 A 与 C，则在之后不会发生任何移动。

2.4 k-Server on a Line

接下来我们将给出一种算法，从较为特殊的情况开始，逐渐一般化，首先考虑一条线上的 k-Server 问题。

2.4.1 Algorithm DC

如果请求发生在 server 的凸包之外，则派遣距离它最近的 server 前往（和贪心一致）；

如果请求发生在两个 server 之间，则两个 servers 将同时前往这一顶点（以相同的速度），先到达一方将进行服务，而此时未到达的一方停止；

注意这里“停止”的含义, 实际上未到达的 server 仍然位于先前的顶点，但是如果之后的请求发生在相同位置，它将继续先前走过的路开始前进，这可以等价地视作 server 所在的这一点与请求发生的顶点之间的边的权重降低了。

继续考虑上述贪心算法中的最坏情况，这一算法可以有效避免这个情况。

2.4.2 Analysis

Theorem 1 *DC 算法是 K-competitive 算法；*

Proof 2.1 *通过 Amortized Analysis 的 Potential Function 方式来证明这一点。*
我们将构造一个势能函数 Φ，它具有如下特点：

假设最优解 OPT 与 DC 算法，对于相同问题同时进行运算，
如果 OPT 将 server 移动了距离 d，Φ 至多增加 kd,
如果 DC 将 server 移动了距离 d，Φ 最少减少 d；
假设 Φ_i 是第 i 次请求发生后势能函数的值；
则 n 次请求发生后势能函数的增量：$\Phi_n - \Phi_0$，最多为 $k * OPT(\sigma) - DC(\sigma)$。

Potential Function 如下：
M_{min} 表示 OPT 的 servers 与 DC 的 servers 之间最小权完美匹配的 cost。
s_i 表示 DC 的第 i 个 server 所在的顶点
$\Sigma = \Sigma_{i<j} d(s_i, s_j)$
则势能函数为：$\Phi = k * M_{min} + \Sigma$

(PS: 别问是怎么构造的，问就是试出来的)

可以依次考虑，一次请求下仅有 OPT 移动了 server/仅有 DC 移动了 server/双方都移动了 server 的情况，可以发现构造的函数是符合之前的要求的。

2.5 k-Server on a Tree

在树图的情况下，考虑如下定义：

一个请求的 **neighbor**: 当一个 server 所在的顶点与请求所在的顶点之间的通路上没有其他 server，则称这个 server 为该请求的 neighbor。

当一个请求发生时，一个请求所有的 neighbor 都以相同的速度向请求派遣（执行方式与之前的 k-Server on a Line 一致）。

同样可以证明这也是一个 K-competitive 算法。

2.6 k-Server on a General Network

定义一个算法的 configuration C 为 K 个 servers 所在顶点的集合。

2.6.1 Work-Function Algorithm (WFA)

定义 Working Function $W(C)$ 表示从初始 configuration C_0 经过请求序列 σ_i, 到最终的 configuration C 的最优 cost。

函数 $argmin()$ 表示使得 () 内函数式达到最小时变量的取值;

则对于请求 r_{i+1}, 该算法将移动 server $s \in C$ 至点 r
$s = argmin_{x \in C} W(C - x + r) + d(x, r)$

可以证明这是一个 (2k-1)-competitive 算法

2.7 k-Server 的相关猜想

1. k-Server 问题存在一个 k-competitive 的在线算法;

2. WFA 本身是一个 k-competitve 的算法（只不过没人给出约束为紧的例子）。