# 近似算法的设计与分类

李华繁

huafan@seu.edu.cn

2016.8

**摘要**

本文以 TSP 和网络设计的一些典型问题为例，介绍了设计近似算法的一些常用方法和技巧，包括贪婪算法、局部搜索、线性规划松弛等。

## 1 方法论简述

现实生活中的许多组合优化问题都是 NP-难的，除非 $P = NP$，它们是没有同时满足下面三个条件的算法：（1）找到全局最优，（2）复杂度是多项式时间，（3）适用于该问题的任何实例。但是这并不是说对这类问题的研究就没有意义可言，相反，这个领域成为了科学家们研究热情最高的领域之一。人们不再执着于求精确的全局最优，而是退而求其次，开始了对近似算法的研究。

### 1.1 近似算法与精确算法的不同

首先，它们处理的问题不同。精确算法主要用来求解可多项式时间可解的问题，而近似算法主要是用来求解 NP-难问题。其次，也是更重要的一点，精确算法和近似算法强调算法性能的不同方面。对于精确算法来讲，研究的重点是算法的运行时间，即算法的效率，主要探讨如何引进设计技巧和数据结构，来减少算法的时间或者空间复杂度。而近似算法的运行时间必须与其性能比（亦称近似比）一同考虑。运行时间与性能比之间的权衡是设计近似算法时需要考虑的一个关键因素。许多近似算法的设计技巧的出发点就是要改进算法的性能比，而同时尽可能减少由此而增加的运行时间。

## 1.2 近似算法的设计角度

在工程领域，有句谚语说，“快速，便宜，可靠。只能从中选二。”类似地，在设计一个 NP-难问题的近似算法时，应该放松上面三点要求中的至少一点，将原来难处理的问题做一个恰当的扰动变换，使得变换后的问题是容易处理的，而且它的最优解与原始问题的最优解又充分接近。一个角度是放松对“任何实例”的要求，而是转而针对该问题的一些特殊实例设计多项式时间算法。还有一个角度是放松对多项式时间求解的要求，而想办法在整个解空间里高效地搜索全局最优。如果允许花上好几分钟乃至几个小时，这种方法往往是效果最好的。但是这种方法有其局限性，在对实时性要求比较高的场合就不适用。

而目前采用最多的方法，是对“一定要找到全局最优”的放松，转而去更快地求解一个“近似解”“比较好的解”或者“可以接受的解”。例如，在应用贪婪策略和局部搜索方法的时候，我们可以将扰动变换作用在问题的目标函数上。而在应用限制和松弛方法的时候，又可以将扰动变换作用在问题的可行域上。这些技巧与设计高效的精确算法所用的技巧（如分治、动态规划、线性规划等）有很大的不同。研究这些设计技巧是近似算法理论的一个重要部分 [25]。

下面给出性能比的概念，它是用来度量算法求得的近似解与问题的最优解之间差距的一种指标。

**定义 1.** 一个优化问题的 $\alpha$-近似算法，是对于该问题所有实例，都能在多项式时间之内求出一个与最优解相差一个因子 $\alpha$ 的解。称 $\alpha$ 是该近似算法的**性能比**（或者**近似比**、**近似因子**）。

对最小化问题，$\alpha > 1$，对最大化问题，$\alpha < 1$。比如，对最大化问题的一个 $\frac{1}{2}$-近似算法，它能在多项式时间之内返回一个至少是最优解一半的近似解。在分析一个近似算法的性能时，通常按如下方法评价：对于最小化问题，首先给出最优解的一个下界，然后把算法的运行结果与这个下界进行比较；而对于最大化问题，先给出一个上界然后把算法的运行结果与这个上界比较。

不同的近似算法通常都是针对具体的问题而设计的，对问题的具体特性依赖很大，因此不太可能存在普适的近似算法（即便对于遗传算法、蚁群优化这类进化算法，也并要根据具体的问题设计合适的算子，因为进化思想只能说是一个框架，几乎不太可能存在一劳永逸的算法）。但是，近似算法的

设计是有章可循的，算法设计中一些有代表性的思想方法，也是近似算法设计思想的主要来源。

# 2 有代表性的设计思想

本节，以**旅行商问题**（Traveling Salesman Problem, TSP）以及一些典型的**网络设计**（Network Design）问题为例来展示近似算法的一些常用设计思想及技巧。关于图论、组合优化，以及近似算法，有一些经典的书籍供参考：[2, 20, 21]。

TSP 是一个著名的 NP-难问题，它可以简单描述为：找一条遍历给定的若干个城市中所有城市一次且仅一次的路径（环游），且路径尽可能短。用图论的方式定义为：有一个图 $G = (V, A)$，其中 $V = \{v_1, v_2, \ldots, v_n\}$ 是 $n$ 个节点，$A = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ 是一个弧集，另外与 $A$ 相关联的有一个非负的价值矩阵（距离矩阵）$C = (c_{ij})$ 表示从节点 $i$ 到达节点 $j$ 的代价，目标是找到一个有最小代价的 Hamiltonian 环。这里只考虑完全图上的对称 TSP（即满足 $c_{ij} = c_{ji}$），且满足三角不等式（即 $c_{i,j}+c_{j,k} \geq c_{i,k}, \forall i, j, k \in V$）。

网络设计的一个典型例子是，给定一个有向图或无向图 $G = (V, E)$，每条边 $e \in E$ 有非负的代价 $c_e$，目标是在 $G$ 中找到一个满足某些设计准则的一个最小代价的子图 $H$。例如，一个简单的问题是，无向图最小生成树（minimum-cost spanning tree）问题；或者在有向图中找一个使得图中任意两点互相可达的弧集，即：最小强连通子图（minimum-cost strongly connected subgraph）问题。这些抽象的模型在网络设计中有大量的实际应用，比如电信和交通网络的设计，超大规模集成电路的设计，等等。

## 2.1 贪婪算法

在贪婪算法中，解是一步一步构造的，在每一步中，基于当前所得的信息，算法所做得决策总是力求找到局部最优。

许多离散优化问题都可以描述或转化为某集合函数（set function）的极值问题。以集合函数的极大化问题为例说明贪婪算法的格式。设 $E = \{e_1, e_2, \ldots, e_n\}$ 有 $n$ 个元素，每个元素 $e_i$ 有一个非负权 $w_i$。$\mathcal{F}$ 是 $E$ 中具有某种性质的子集族，满足以下两个性质：(1) $\emptyset \in \mathcal{F}$，(2) 若 $A \in \mathcal{F}$ 且 $Y \subseteq A$，有 $Y \in \mathcal{F}$。$E$ 中任一子集 $S$ 的权，定义为 $W(S) = \sum_{e_i \in S} w_i$。问题是求 $\mathcal{F}$

中权最大的集合，即：

$$\begin{aligned} \underset{S}{\text{maximize}} \quad & W(S) \\ \text{s.t.} \quad & S \in \mathcal{F} \end{aligned} \tag{1}$$

给出如下的算法框架：

**Algorithm 1 极大化集合权**

1: 将 $E$ 中元素根据权值由大到小排列，不失一般性，设 $w_1 \geq w_2 \geq \ldots \geq w_n$，置 $T = \emptyset, i = 1$；
2: 若 $T \cup \{e_i\} \in \mathcal{F}$，置 $T \leftarrow T \cup \{e_i\}$；
3: 若 $i = n$，终止，输出 $T$；否则置 $i \leftarrow i + 1$，返回步骤 2。

值得一提的是，贪婪算法的理论基础是拟阵（Matroid），它是一种很漂亮的理论，在确定贪婪法什么时候能产生最优解时非常有用（虽然目前这种理论并没有覆盖贪婪法所适用的所有情况，但是它正处于发展中，相信会越来越完备）。对于某些符合拟阵性质的问题，使用贪婪算法能够求得全局最优。比如经典的最小生成树问题，已知的最早的算法——Kruskal 算法和 Prim 算法，都是典型的贪婪算法 [13, 16]，而且它们被证明，对任何有任意边代价的连通图都能找到最小生成树 [18]。

### 2.1.1 最近邻法

回到 TSP 问题。一个最直观也非常简单的贪婪算法是最近邻法（Nearest Neighbor），其思想就是每次访问最近的城市。

**Algorithm 2 最近邻算法**

1: 随机选择一个城市。
2: 找到下一个没访问过的最近的城市，选择它。
3: 判断是否还有没访问过的城市，如果是，返回步骤 2。
4: 回到第一个城市。

不难得到，最近邻法的时间复杂度为 $O(n^2)$。关于最近邻法，目前已知其最好性能比是 $\frac{1}{2}(\lceil \log n \rceil + 1)$[17]。虽然最近邻算法本身很简单，但是要证明其性能比却不容易，往往要构造一些不等式，这些不等式来源于对问题本身结构和特性的观察。所以有必要对一些简单的算法分析其性能，这有利于

我们更好地认识问题，从而为设计复杂而高效的算法奠定基础。下面，我们重现对最近邻算法的性能比分析。

设对任意一个输入实例 $I$，最近邻法得到的解和最优解对应的路径权值分别为 $NN(I)$ 和 $OPT(I)$，有：

**定理 1.** 对 $n$ 个城市的 $TSP$ 问题，有：

$$\frac{NN(I)}{OPT(I)} \leq \frac{1}{2}\left(\lceil \log n \rceil + 1\right).$$

**证**. 不失一般性，设最近邻算法选择的所有边是 $l_1 \geq l_2 \geq \ldots \geq l_n$（注意，这不一定是算法运行得到的边的长度序列）。然后，给节点编号：由于最近邻算法每一次迭代加入一个节点，所以它所得到的环游可以看做是有向的，我们将长度为 $l_i$ 的弧的起点（尾）编号为 $i$。这种编号方式是不重不漏的（最近邻算法求出的所有的边被编号，图中所有的节点被编号）。

接下来，我们考察节点编号为 $1, 2, \ldots, 2k$ 的这些节点（$2k \leq n$）。它们可以构成原图的一个子图，设它们所构成的完全图是 $H$。如果我们按原图中最佳环游访问节点的顺序连接这 $2k$ 个节点，就可以得到一个该子图的环游，设为 $T_k$。根据三角不等式不难证明 $\mathcal{L}(T_k) \leq OPT(I)$，其中 $\mathcal{L}(T_k)$ 表示路径 $T_k$ 的长度。设 $(i,j) \in T_k$ 是路径 $T_k$ 上的一条边（$1 \leq i, j \leq 2k, i \neq j$），我们知道，在最近邻算法中，如果节点 $i$ 比 $j$ 先加入，那么一定有 $l_i \leq c_{i,j}$；同样地，如果 $j$ 比 $i$ 先加入，那么一定有 $l_j \leq c_{j,i}$，由于 $c_{i,j} = c_{j,i}$，所以得到 $c_{i,j} \geq \min(l_i, l_j)$。于是，有：

$$\mathcal{L}(T_k) \geq \sum_{(i,j)\in T_k} \min(l_i, l_j) \tag{2}$$

这就建立了 $T_k$ 和最近邻算法所得解之间的联系，上面这个不等式将 $T_k$ 中的每条边映射为 $l_1, l_2, \ldots, l_{2k}$ 中的 $2k$ 条边，注意映射后的每条边最多可能出现两次。又由于 $l_1 \geq l_2 \geq \ldots l_{2k}$，所以最小的极端情况是 $l_{2k}, l_{2k-1}, \ldots, l_{k+1}$ 这 $k$ 条边每条被映射两次，因而有：

$$\sum_{(i,j)\in T_k} \min(l_i, l_j) \geq 2 \sum_{i=k+1}^{2k} l_i \tag{3}$$

结合式子 (2) 和 $\mathcal{L}(T_k) \leq OPT(I)$，即得到：

$$OPT(I) \geq 2 \sum_{i=k+1}^{2k} l_i \tag{4}$$

同样地，如果考察的是编号为 $1,2,\ldots,n$ 这些节点, 此时，$H=G$，最终可以得到：

$$OPT(I) \geq 2 \sum_{i=\lceil \frac{n}{2} \rceil+1}^{n} l_i \tag{5}$$

在式子 (4) 中分别令 $k=1,2,2^2,\ldots,2^s$（$2^{s+1}$ 是不超过 $n$ 的最大整数），得：

$$\begin{aligned} &OPT(I) \geq 2 \cdot l_2 \\ &OPT(I) \geq 2 \cdot (l_3+l_4) \\ &OPT(I) \geq 2 \cdot (l_5+l_6+l_7+l_8) \\ &\vdots \\ &OPT(I) \geq 2 \cdot (l_{2^s+1}+l_{2^s+2}+\ldots+l_{2^{s+1}}) \end{aligned} \tag{6}$$

将上面 $(s+1)$ 个不等式相加，得：

$$(s+1)OPT(I) \geq 2 \cdot (l_2+l_3+l_4\ldots+l_{2^{s+1}}) \tag{7}$$

由于 $2^{s+1} \geq \lceil \frac{n}{2} \rceil$，将不等式 (7) 与不等式 (5) 相加得：

$$(s+2)OPT(I) \geq 2 \cdot (l_2+l_3+l_4\ldots+l_n) \tag{8}$$

根据三角不等式以及 $c_{i,j}=c_{j,i}$，易得：$OPT(I) \geq 2 \cdot l_1$，于是，

$$(s+3)OPT(I) \geq 2 \cdot (l_1+l_2+l_3+l_4\ldots+l_n) \tag{9}$$

由于 $2^{s+1}$ 是不超过 $n$ 的最大整数，所以 $s+1=\lceil \log n \rceil - 1$，于是上式等价于：

$$(\lceil \log n \rceil + 1)OPT(I) \geq 2 \cdot (l_1+l_2+\ldots+l_n) = 2NN(I) \tag{10}$$

至此，证明结束。 □

可见，即使是像最近邻法这样如此简单的算法，通过挖掘和分析，我们也能够较好地了解它的性能。值得一提的是，其实还有“正宗”的贪婪算法，即基于本节一开始给出的框架，基本思想是：首先将完全图的所有边从小到大排序，然后依次选择边构造路径，满足所构造的路径不构成环且所有节点的度不超过 2，直到构造出一个环游。已经证明，这种算法和最近邻算法有类似的性能比。

### 2.1.2 最近插入法

最近插入法 (Nearest Insertion) 也是一种基于贪婪思想的启发式算法。其基本思想是对 $n$ 个城市中的某 $k$ 个城市所构成的一个子环游 $S$，陆续加入一个不在 $S$ 中，但距离 $S$ 最近的城市，直到 $S$ 包含所有 $n$ 个城市为止。如何定义一个不在子环游中的点与该子环游的距离？用该点与子环游中所有点的最小距离表征。即：对某个节点 $i \in V \setminus S$，其到 $S$ 的距离定义为：

$$d(i,S) = \min_{j \in S} c_{i,j}.$$

**Algorithm 3 最近插入法**

1: 选择最短的边，设最短边的两个端点为 $i, j$，在 $i, j$ 上构造一个子环游。用符号 $S$ 表示该子环游。

2: 找到不在 $S$ 中距离 $S$ 最近的点 $i^*$：$i^* = \arg\min\{d(i,S), i \in V \setminus S\}$。

3: 设 $j^*$ 为 $S$ 中与 $i^*$ 最近的点，在 $S$ 中与 $j^*$ 相邻的两个节点分别是 $j_1, j_2$。按下面的规则决定 $i^*$ 插入的位置：如果 $c_{i^*,j} + c_{i^*,j_1} - c_{j^*,j_1} \leq c_{i^*,j} + c_{i^*,j_2} - c_{j^*,j_2}$，则将 $i^*$ 插入到 $(j^*, j_1)$ 之间，否则将 $i^*$ 插入到 $(j^*, j_2)$ 之间。置 $S \leftarrow S \cup \{i^*\}$。

4: 如果 $S$ 已经包含了所有的节点，结束；否则返回步骤 2。

时间复杂度分析：步骤 1（第 1 次迭代）中要计算最近点对，暴力枚举的话是 $O(n^2)$；步骤 2，第 2 次迭代开始时 $S$ 中有两个点，第 $k$ 次迭代时 $S$ 中有 $(k)$ 个点，$S$ 之外有 $(n-k)$ 个点，对 $S$ 之外的每个点计算其与 $S$ 的距离须要比较 $k$ 次，一共须要比较 $(n-k)k$ 次。注意在算法实现时可以做一步很大的优化：计算每个点到 $S$ 的距离没有必要每次都重新计算 $d(i,S)$，往 $S$ 中加入 $i^*$ 后只要更新每个 $S$ 之外的点与 $S$ 的距离即可，这一步可以 $O(1)$ 完成，更新方法是 $d(i,S) = \min\{d(i,S), c_{i,i^*}\}, \forall i \in V \setminus S$，这样可以避免许多重复的计算，只要比较 $(n-k)$ 个点与 $S$ 的距离即可确定 $i^*$。另外，步骤 3 可以 $O(1)$ 完成。一共有 $n$ 次迭代，所以整个算法的复杂度为：

$$O(n^2) + O\left(\sum_{k=1}^{n} n - k\right) = O(n^2).$$

对该算法的近似性能分析的桥梁是最小生成树。通过如下引理给出 TSP 的最佳环游与最小生成树之间的关系：

**引理 1.** 对 *TSP* 问题的任何输入实例 $I$，最优环游的代价至少是图中最小生成树的代价。即最小生成树的代价是 *TSP* 最优解的一个下界。

**证**. 证明很容易。对任意 $n \geq 2$ 的输入，设我们已经有了 TSP 的一个最优环游。显然，如果删除这个最优环游的任意一条边，就得到了输入实例的一个生成树，而且它的代价小于最优环游。另一方面，这个生成树的代价一定不小于最小生成树的代价，所以最优环游的代价一定不小于最小生成树的代价。 □

设对任意一个输入实例 $I$，记最近插入法得到的解对应的路径代价为 $NI(I)$。下面我们证明最近插入算法的性能比：

**定理 2.** 对任意规模的对称 *TSP* 问题，有：

$$\frac{NI(I)}{OPT(I)} \leq 2$$

**证**. 证明的关键在于建立算法每一步的计算与最小生成树的关系。注意到算法的步骤 2 的计算方式与计算最小生成树的 Prim 算法非常相似，不同的是，在 Prim 算法中，是直接把找到的边加入正在构建的生成树中，而最近插入法并不是把边 $(i^*, j^*)$ 加入，而是把点 $i^*$ 插入子环游中，扩成一个新的子环游。不失一般性，做如下符号定义：设 $i_1^*, j_1^*$ 是算法在执行步骤 1 后找到的两个点，在这两个点上形成的子环游记为 $S_1$；之后第 $k(k = 2, 3, \ldots, n)$ 次迭代时求得的 $i^*$ 记为 $i_k^*$，$j^*$ 记为 $j_k^*$，$j_1, j_2$ 分别记为 $j_1^{(k)}, j_2^{(k)}$，插入 $i_k^*$ 后的子环游记为 $S_k$。

注意，如果每次迭代不是扩充环游，而是加入边 $(i_k^*, j_k^*)$，那么我们最终将得到最小生成树，因为操作是与 Prim 算法一致的。记输入实例 $I$ 的最小生成树对应的权值为 $MST(I)$，则 $MST(I) = \sum_{i=1}^{n} c_{i_k^*, j_k^*}$。在第 $k$ 次迭代中，相比子环游 $S_{k-1}$，$S_k$ 的增量为

$$\Delta_k = \min\left\{c_{i_k^*, j_k^*} + c_{i_k^*, j_1^{(k)}} - c_{j_k^*, j_1^{(k)}},\ \ c_{i_k^*, j_k^*} + c_{i_k^*, j_2^{(k)}} - c_{j_k^*, j_2^{(k)}}\right\} \tag{11}$$

其中的两个值分别对应将 $i_k^*$ 插入 $j_k^*, j_1^{(k)}$ 之间、$j_k^*, j_2^{(k)}$ 之间子环游的增量。这也是最近插入法决定插入哪两个点之间的依据。而根据三角不等式，无论将 $i_k^*$ 插入哪两点之间，都有 $c_{i_k^*, j_1^{(k)}} - c_{j_k^*, j_1^{(k)}} \leq c_{i_k^*, j_k^*}$，以及 $c_{i_k^*, j_2^{(k)}} - c_{j_k^*, j_2^{(k)}} \leq c_{i_k^*, j_k^*}$，从而有 $\Delta_k \leq 2 \cdot c_{i_k^*, j_k^*}$。另外，在第 1 次迭代中（步骤 1），从空集开始构造出一个有两个点的环，所以增量恰好是 $\Delta_1 = 2 \cdot c_{i_1^*, j_1^*}$。于是，将最近

插入算法的所有迭代步的增量相加，就得到最终环游的代价：

$$NI(I) = \sum_{k=1}^{n} \Delta_k \leq \sum_{k=1}^{n} 2 \cdot c_{i_k^*, j_k^*} = 2 \cdot MST(I) \tag{12}$$

由引理 1知，$MST(I) \leq OPT(I)$，从而 $NI(I) \leq 2 \cdot OPT(I)$。证明完成。 □

### 2.1.3 生成树加倍法

生成树加倍法（Double Spanning-Tree, DST）的灵感大概也是来源于最小生成树与最优 TSP 环游之间的关系。

**Algorithm 4 生成树加倍法**

1: 计算出图 $G$ 的一个最小生成树 MST。
2: 将 MST 的每条边复制一次，得到一个多重图（欧拉图）。
3: 在多重图上找一个欧拉环，记为 $\mathcal{T}$。
4: 删去 $\mathcal{T}$ 中重复出现的节点，输出所得到的环游，结束。

时间复杂度分析：步骤 1，计算最小生成树，由于图是完全图，非常稠密（$m = \frac{1}{2}n(n-1)$），用 Prim 算法较好，数据结构采用邻接矩阵，可以 $O(n^2)$ 计算出一个最小生成树；步骤 2，遍历每个节点一次即可，$O(n)$；步骤 3 和步骤 4 都可以 $O(m) = O(n^2)$ 完成（找欧拉环也就是常说的“一笔画”问题）。整个算法的复杂度是 $O(n^2)$。

设对任意一个输入实例 $I$，记生成树加倍法得到的解对应的路径代价为 $DST(I)$，最优环游的代价为 $OPT(I)$。下面我们证明生成树加倍法的性能比：

**定理 3.** 对任意规模的对称 *TSP* 问题，有：

$$\frac{DST(I)}{OPT(I)} \leq 2$$

**证**. 记最小生成树的代价为 $MST(I)$，首先，由引理 1，$MST(I) \leq OPT(I)$。由于 $\mathcal{T}$ 将最小生成树上所有的边都复制了一次，所以 $\text{cost}(\mathcal{T}) = 2 \cdot MST(I)$。在步骤 4 中，由于删除了 $\mathcal{T}$ 中重复出现的节点（相当于“走捷径”）来构造 TSP 环游，根据三角不等式，必有 $DST(I) \leq cost(\mathcal{T})$。所以，$DST(I) \leq 2 \cdot MST(I) \leq 2 \cdot OPT(I)$。 □

上面这个算法还有改进的空间，Christofides(1976) 在此基础上将近似比降到了 1.5 倍 [22]，而且这个界是紧的，这是非常好的一个结果了。改进的算法是这样的：首先，计算出一个最小生成树 MST；然后，在 MST 中有奇度数的节点上求一个最小权完美匹配 $M$，可以证明 $M$ 的边代价不会超过 $\frac{1}{2}OPT(I)$；将 $M$ 与 MST 合并就能得到一个连通图，且其中所有的节点度数为偶数，因此一定存在欧拉环。同样地，找到欧拉环后，删除所有重复地节点，就能得到一个不超过最优解 $\frac{3}{2}$ 倍的 TSP 环游。

以上只列举了求解 TSP 问题最典型的几种基于贪婪思想的近似算法，它们的变种还有很多很多。但是我们可以发现一些共性：基本上都有一个中间桥梁，这个中间桥梁往往是某种较为简单的贪婪算法，它们能够为问题的最优解提供一个不错的下界，并且能够与所设计（或者分析）的近似算法建立联系，从而可以较好地估计近似算法的性能比。

## 2.2 局部搜索

局部搜索（Local Search）也常称为“邻域搜索”。它是一种古老的优化方法，其思想也十分自然：从给定问题的一个初始可行解开始，在其邻域内进行搜索，如果能找到更好地解，就保存下来，直到解无法优化为止，这时就达到了一个局部最优解。有时可能可以证明这个局部最优解与全局最优解之间存在一定的近似关系。注意与其他近似算法设计技术不同的是，局部搜索算法最直接的实现并不是多项式时间的，通常对局部的变换加入一些限制条件，以确保在每一步迭代中能够进行足够的优化，从而在多项式时间之内找到局部最优。局部搜索算法的框架如下：

**Algorithm 5 局部搜索框架**

1: 取问题的一个初始可行解 $\mathbf{x}_0$ 和其邻域 $N(\mathbf{x}_0)$。
2: 在邻域 $N(\mathbf{x}_0)$ 内搜索，如果找到比 $\mathbf{x}_0$ 更好的可行解 $\mathbf{x}$，则置 $\mathbf{x}_0 = \mathbf{x}$，重复步骤 1；否则 $\mathbf{x_0}$ 是局部最优解，结束。

在局部搜索的框架中，有两点需要说明。首先需要找出问题的一个初始可行解，而这对许多优化问题来说并不容易，比如在 2016 年的华为软件精英挑战赛中 [10]，赛题是一个 NP-难问题：在有向图中，给定起点和终点，找一条经过指定必经点集的最短路径，且任何一个节点仅能访问一次。对这个问题，很难构造初始可行解，甚至无法在多项式时间之内判断是否存在一个可行解。其次，如何确定可行解的邻域？对不同的问题，邻域的结构是不

同的，即使对相同的问题，邻域的大小也往往有很大的变化范围。邻域大的，求出的局部极值更精确，但计算量就可能增大。但是无论邻域的结构和大小怎样，都应该满足的是，在邻域内的搜索是多项式时间可以完成的。

比如，线性规划的单纯型算法就属于局部搜索算法。一个基可行解 $\mathbf{x}$ 的邻域，是与 $\mathbf{x}$ 恰好有一个基变量不同的基可行解组成（在凸多面体上，是所有与某个顶点相邻的顶点），并且，这样搜索出的局部最优也是全局最优。又如，无向图中的最大权生成树，一个可行解就是该图的一颗生成树。一颗生成树 $T$ 的领域，就是与 $T$ 恰有一条边不同的所有生成树的集合（其局部极值也是全局极值）。

对于 TSP 问题，局部搜索也称为“环路优化”。首先讨论 TSP 的可行解的邻域。TSP 问题的可行解就是过每个节点一次且仅一次的环游 $C$。显然，由可行解 $C$ 可以得到另一个可行解 $C'$，且 $C'$ 与 $C$ 至少有两条不同的边。因此，对给定的一个可行解 $C$，定义 $C$ 的邻域为所有这样一些可行解 $C'$：$C'$ 与 $C$ 最多有 $k$ 条不同的边，其中 $k \geq 2$。所以 $k$ 越大，$C$ 的邻域也越大；当 $k = n$ 时，$C$ 的邻域中就包含最优解，此时邻域中的局部最优解也是全局最优解。但是邻域越大，解空间规模就越大，求局部最优也越困难。所以必须选取合适的 $k$ 值，以在求解精度和时间复杂度之间做一个较好的权衡。

### 2.2.1 $k$-opt

$k$-opt 操作也叫 $k$-exchange。我们先来讨论 $k = 2$ 和 $k = 3$ 的情形。2-opt 和 3-opt 也是局部搜索算法中最著名的两种技术。2-opt 由 Croes 于 1958 年首先提出 [3]，基本操作方法是在一个可行 TSP 环路中删掉两条边，这样就把环游分成了两条路径，然后将这两条路径重组成与原来不同的新的环路。2-opt 其实只有一种情况，如图 1所示，注意这只是示意图。只有当新的环游比原来的短时，才执行交换。如果发现对某一个环路，其所有两条边的组合都无法改进该环路，那么就达到了局部最优，可以结束算法。

类似地，3-opt 操作 [14] 是打破原环路的三条边，对剩下的三条路径进行重组，3-opt 操作有两种情况，图 2描绘了这两种可能的情况。由于 3-opt 的邻域包含了 2-opt 邻域，所以如果一个环路是 3-opt 最优的，那么它也一定是 2-opt 最优的。

时间复杂度分析：2-opt 和 3-opt 分别是 $O(n^2)$ 和 $O(n^3)$，而 $k$-opt 则是 $O(n^k)$，但是在具体实现上，使用不同的数据结构将大大影响算法的运行

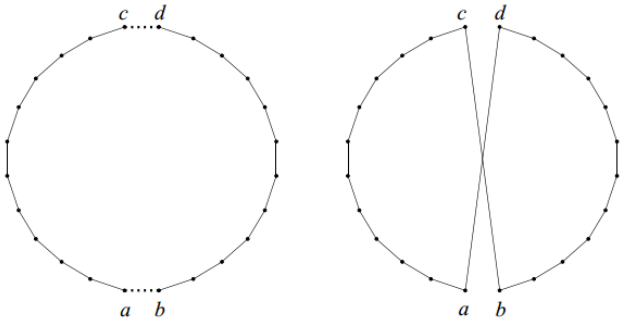


图 1: 一次 2-opt 操作

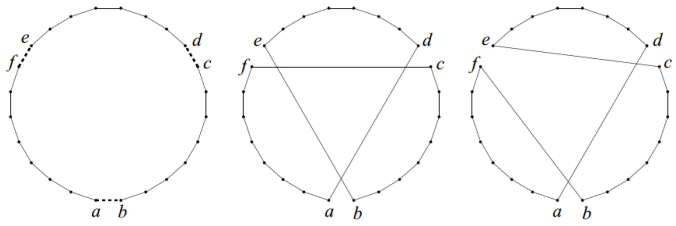


图 2: 两种可能的 3-opt 操作

效率。

$k$-opt 的理论近似比并不优秀。Chandra, Karloff 和 Tovey 于 1994 年证明，2-opt 的最好性能保证至少是 $\frac{1}{4}\sqrt{n}$，3-opt 的性能保证至少是 $\frac{1}{4}n^{1/6}$，$k$-opt 则至少是 $\frac{1}{4}n^{1/(2k)}$ [11]。目前，还没有得到 2-opt 和 3-opt 的紧的界。但是如果把条件限制为几何 TSP，即所有城市是 $\mathbb{R}^d$ 维空间中的点，它们之间的距离用 $L^d$ 范数计算，那么情况会好一些：虽然即便在欧式空间，性能比也会以 $\Theta(\frac{\log n}{\log\log n})$ 的速度增长，但是所有 $k$-opt 算法的最坏性能比是 $O(\log n)$。Chandra 等人同样证明了在几何空间中，最差的 2-opt 环游的期望近似比是一个常数。

在实际测试中，$k$-opt 的性能很优越。对邻域搜索算法近似性能评估要通过大量的实验来进行，常用的一个评估准则是将结果与 Held-Karp (HK) 下界 (Held-Karp lower bound) 进行比较。这个下界其实就是 TSP 整数规划模型的线性规划松弛解 [12]（可以多项式时间计算出来）。实验表明，在平均意义上，HK 下界大约比最优环游的长度小 0.8%；注意，在理论上的结果是，这个下界只有最优环游长度的 2/3。实际情况下的测试显示，通常 2-opt 算法得到的解不会超过 Held-Karp 下界的 5%，而 3-opt 则不会超过 HK 下界的 3%，这都是很好的结果了。

基于 $k$-opt 的思想，Lin 和 Kernighan 设计了一个与 Held-Karp 下界只在 2% 以内的算法，在大量实测中的性能表现非常好，被公认为是 TSP 局部搜索算法中性能最好的算法，它目前仍保持着 TSP 标准测试集中好几个用例的世界纪录 [7]。而且其复杂度只有 $O(n^{2.2})$，关于该算法的高效实现请参见文献 [8]。

## 2.3 线性规划松弛

松弛技术是获得最优解下界的一种好方法，大多数经典的松弛主要通过线性规划来实现。线性规划松弛技术是设计近似算法的一种非常有用的算法。整数线性规划是 NP-难的，但如果放松整数的约束，其松弛问题就是线性规划，而线性规划可以多项式时间求解。通过解整数线性规划的松弛问题，对松弛问题的解进行处理，从而可以得到整数规划的近似解，或者解松弛问题的对偶来得到原整数规划的可行解。

基于线性规划的近似算法设计框架如下：

上面的基本步骤中，最重要的步骤是第 4 步。这一步常常使用的有两种主要策略：（1）凑整技术（Rounding），（2）原始——对偶方法（Primal-

**Algorithm 6 基于线性规划松弛的近似算法**

1: 将实际带求解问题形式化为整数规划（IP）模型。
2: 利用松弛技术，将 IP 问题松弛为 LP 问题。
3: 采用某种策略，将 LP 问题的解转化为整数解。
4: 将整数解重新形式化为 IP 问题的解，得到原问题的解。

Dual）。下面，我们以带权顶点顶点覆盖问题为例，分别解释这两种技术。首先，描述一下这个问题。

给定一个赋权无向图 $G = (V, E)$，每个顶点 $v \in V$ 都有一个正的权值 $w(v)$。图 $G$ 的一个**顶点覆盖**是 $V$ 的一个子集 $V'$，其中的点关联了 $G$ 中所有的边（即：对任意一条边 $(u, v) \in E$，其两个端点 $u, v$ 至少有一个在 $V'$ 中）。很自然地，一个顶点覆盖 $V'$ 的**权**就是该顶点覆盖中所有点的权值之和，即 $w(C) = \sum_{v \in V'} w(v)$。**最小权顶点覆盖问题**（minimum-weight vertex-cover problem, WVC），就是找到图中权最小的顶点覆盖。最小权顶点覆盖问题是 NP-难的 [19]。

为图 $G = (V, E)$ 中每个顶点 $v \in V$ 建立一个 0-1 决策变量 $x(v)$，即 $x(v) \in \{0, 1\}$，$x(v) = 0$ 表示顶点 $v$ 不在顶点覆盖中，$x(v) = 1$ 则表示 $v$ 在顶点覆盖中。于是，最小权顶点覆盖问题的整数规划模型是：

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} w(v)x(v) \\ \text{s.t.} \quad & x(u) + x(v) \geq 1, \quad \forall (u, v) \in E \\ & x(v) \in \{0, 1\}, \quad \forall v \in V \end{aligned} \tag{13}$$

松弛对决策变量的整数约束条件，我们得到相应的线性规划问题 LP：

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} w(v)x(v) \\ \text{s.t.} \quad & x(u) + x(v) \geq 1, \quad \forall (u, v) \in E \\ & 0 \leq x(v) \leq 1, \quad \forall v \in V \end{aligned} \tag{14}$$

记原整数规划模型的最优解是 $OPT(I)$，线性规划模型的最优解是 $OPT_{LP}(I)$，则 $OPT_{LP}(I)$ 是 $OPT(I)$ 的一个下界。

### 2.3.1 凑整方法

我们给出基于凑整（Rounding）的算法，这个算法的思想也很自然：将线性规划的最优解向量中不小于 $\frac{1}{2}$ 的凑整为 1，小于 $\frac{1}{2}$ 的则凑整为 0。Rounding-MinWVC 算法的时间复杂度主要取决于第 2 行的求线性规划模

**Algorithm 7** ROUNDING-MINWVC

```
1: 置 C ← ∅
2: 求线性规划的最优解 x*
3: for each v ∈ V do
4:   if x*(v) ≥ 1/2 then
5:     C ← C ∪ {v}
6:   end if
7: end for
8: return C
```

型的复杂度，而这一步可以多项式时间求得最优解。对于其近似性能比，有如下命题：

**定理 4.** *Rounding-MinWVC* 算法是最小带权顶点覆盖问题的多项式时间 2-近似算法。

**证**. 首先说明 Rounding-MinWVC 算法得到的点集 $C$ 是一个顶点覆盖。因为对 $G$ 的任意一条边 $(u,v) \in E$，按 LP 约束 $x(u)+x(v) \geq 1$，可知 $x(u)$ 和 $x(v)$ 至少有一个不小于 $\frac{1}{2}$，意味着 $u,v$ 至少有一个被加入了 $C$，也即图 $G$ 的任意一条边被 $C$ 覆盖。接下来说明近似比。记 $C$ 的权为 $A(I)$。

考虑 LP 模型的最优解，有：

$$\begin{aligned} OPT_{LP}(I) &= \sum_{v \in V} w(v)x^*(v) \\ &= \sum_{v \in V: x^*(v) \geq 1/2} w(v)x^*(v) \\ &\geq \sum_{v \in V: x^*(v) \geq 1/2} w(v)\frac{1}{2} \\ &= \sum_{v \in C} w(v) \cdot \frac{1}{2} \\ &= \frac{1}{2}w(C) = \frac{1}{2}A(I) \end{aligned} \tag{15}$$

又由于 $OPT_{LP}(I) \leq OPT(I)$，所以，$A(I) \leq 2 \cdot OPT_{LP}(I) \leq 2 \cdot OPT(I)$。 □

### 2.3.2 原始——对偶方法

基于原始——对偶方法的线性规划松弛能够更加有效地获得一个近似解。其主要思想是：任何一个对偶可行解都是最小化原始问题的下界，利用互补松弛性条件可以找到一系列成对的原始、对偶近似解，直到所得原始近似解满足要求为止。先简单介绍一下相关概念。

对于极小化问题，线性规划的一个原始问题（LP）：

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, 2, \ldots, m \\ & x_j \geq 0, \quad j = 1, 2, \ldots, n \end{aligned} \tag{16}$$

其对偶问题（DLP）是：

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{m} b_i y_i \\ \text{s.t.} \quad & \sum_{i=1}^{m} a_{ij} y_i \leq c_j, \quad j = 1, 2, \ldots, n \\ & y_i \geq 0, \quad i = 1, 2, \ldots, m \end{aligned} \tag{17}$$

利用弱对偶定理 [26]，可以找到原始问题最优解的一个下界：

**定理 5** (**弱对偶定理**). 对于每一个原始可行解 $\mathbf{x}$ 以及每一个对偶可行解 $\mathbf{y}$，有 $\sum_i y_i b_i \leq \sum_j c_j x_j$。

简单回顾一下线性规划的原始——对偶算法：从对偶可行初始解和原始非可行初始解开始，迭代地修改对偶可行解，并在满足互补松弛性的前提下修改原始解，当可行性和互补松弛性同时满足时，就得到了原始问题和对偶问题的最优解。

**定理 6** (**互补松弛性定理**). 设 $\mathbf{x}$ 和 $\mathbf{y}$ 分别是 *LP* 和 *DLP* 的可行解。那么当且仅当以下的两个条件同时满足时，$\mathbf{x}$ 和 $\mathbf{y}$ 为 *LP* 和 *DLP* 的最优解：

**原始条件**：$x_j = 0$，或 $\sum_{i=1}^{m} a_{ij} y_i = c_j, \forall j = 1, \ldots, n$

**对偶条件**：$y_i = 0$，或 $\sum_{j=1}^{n} a_{ij}x_j = b_i, \forall i = 1, \ldots, m$

而基于原始——对偶方法设计近似算法时，我们不再要求互补松弛性必须满足，而放松对这个条件的要求，得到**松弛的互补松弛性条件**：

**定义 2** ($(\alpha, \beta)$ **互补松弛性条件**). 设 $\mathbf{x}$ 和 $\mathbf{y}$ 分别是 $LP$ 和 $DLP$ 的可行解。$\alpha \geq 1, \beta \geq 1$：

$\alpha$-**松弛原始条件**：$x_j = 0$，或 $\frac{c_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij}y_i \leq c_j, \forall j = 1, \ldots, n$

$\beta$-**松弛对偶条件**：$y_i = 0$，或 $b_i \leq \sum_{j=1}^{n} a_{ij}x_j \leq \beta \cdot b_i, \forall i = 1, \ldots, m$

我们从一个原始——对偶序对 $(\mathbf{x}, \mathbf{y})$ 开始，其中 $\mathbf{x}$ 是原始变量，可能不是可行解，$\mathbf{y}$ 是对偶变量，可能不是对偶问题的最优解。在算法的每一次迭代，我们尝试让 $\mathbf{y}$ "更优"，且 $\mathbf{x}$ "更可行"，直到 $\mathbf{x}$ 是可行解为止。算法框架如下：

**Algorithm 8 基于原始——对偶的线性规划松弛近似算法**

1: 初始化：对偶可行解 $\mathbf{y}$ 和原始非可行解 $\mathbf{x}$；
2: 修改对偶解和原始解：在满足松弛的互补松弛性条件的前提下，提高对偶解 $\mathbf{y}$ 的目标函数值，并在保证原始解 $\mathbf{x}$ 的整数性前提下提高其可行性。
3: 原始解是否可行？是则输出原始解，得到原整数规划问题的一个近似解，结束；否则返回步骤 2。

**定理 7.** 满足 $(\alpha, \beta)$ 互补松弛性条件的原始——对偶近似算法的近似比不超过 $\alpha \cdot \beta$。

**证**.

$$\begin{aligned}
\sum_{j=1}^{n} c_j x_j &\leq \sum_{j=1}^{n} \left( \alpha \cdot \sum_{i=1}^{m} a_{ij}y_j \right) x_j \\
&= \alpha \sum_{j=1}^{n} \left( \sum_{i=1}^{m} a_{ij}y_j \right) x_j \\
&= \alpha \sum_{i=1}^{m} \left( \sum_{j=1}^{n} a_{ij}x_j \right) y_i \\
&\leq \alpha \cdot \beta \sum_{i=1}^{m} b_i y_i
\end{aligned} \tag{18}$$

再根据弱对偶定理，即可得证。 □

对于顶点覆盖问题，其线性规划松弛的 DLP 形式如下：

$$\begin{aligned} \text{maximize} \quad & \sum_{(u,v)\in E} y_{uv} \\ \text{s.t.} \quad & \sum_{(u,v)\in E} y_{uv} \le w(v), \quad \forall v \in V \\ & y_{uv} \ge 0, \quad \forall (u,v) \in E \end{aligned} \tag{19}$$

显然，$y_{ij}$ 的所有分量都为 0 所对应的解是 DLP 的一个可行解。算法就从该对偶可行解和原始非可行解（如 $\mathbf{x} = \mathbf{0}$）开始迭代：

**Algorithm 9** PrimalDual-MinWVC

初始化。置 $C \leftarrow \emptyset$；设置对偶初始可行解：对所有的 $(u,v) \in E$，置 $y_{uv} \leftarrow 0$；设置原始初始非可行解：对所有的 $v \in V$，置 $x(v) \leftarrow 0$；
**while** $C$ 不是 $G$ 的顶点覆盖 **do**
  取一条未被 $C$ 覆盖的边 $(u,v)$；
  增加 $y_{uv}$，直到它不小于其某个端点的权值，即 $y_{uv} = w(v)$ 或者 $y_{uv} = w(u)$；
  **if** $y_{uv} = w(v)$ **then**
    $x(v) \leftarrow 1$；
    $C \leftarrow C \cup \{v\}$；
  **else**
    $x(u) \leftarrow 1$；
    $C \leftarrow C \cup \{u\}$；
  **end if**
**end while**
**return** $C$

下面证明上面算法的近似性能比：

**定理 8.** PrimalDual-MinWVC 算法是最小权顶点覆盖问题的一个 2-近似算法。

**证**. 首先，根据算法的循环终止条件知 $C$ 一定是图 $G$ 的一个顶点覆盖。注

意到对每个 $v \in C$，都有 $\sum_{(u,v)\in E} y_{uv} = w(v)$，因此，有：

$$\begin{aligned} w(C) &= \sum_{v\in C} w(v) = \sum_{v\in C} \sum_{(u,v)\in E} y_{uv} \\ &\leq \sum_{v\in V} \sum_{(u,v)\in E} y_{uv} \\ &= 2 \sum_{(u,v)\in E} y_{uv} \\ &\leq 2 \cdot OPT_{LP}(I) \\ &\leq 2 \cdot OPT(I) \end{aligned} \tag{20}$$

第一个不等式是因为 $C$ 是 $V$ 的子集，接下来的等式是因为 $E$ 中的每条边被计算了两次，第二个不等式是根据弱对偶定理，最后一个不等式则是整数线性规划和其线性规划松弛的关系。证明结束。 □

## 2.4 其他方法和分类

除了上面介绍的经典近似算法设计方法外，还有一些其他的方法和技巧。

### 2.4.1 计算几何

许多网络设计、路径规划的实际问题，如 TSP、Steiner 树问题、三角剖分等 NP-难问题，它们都有明显的几何特性，因此需要一些特别的算法设计技术，并辅以高效的数据结构——这就是**计算几何学**。计算几何学（Computational Geometry）是在二十世纪七十年代末从算法设计与分析中孕育而生的一个领域，是计算机科学的一个分支，它专门研究那些用来解决几何问题的算法。在现代工程与数学中，计算机图形学、机器人、VLSI（大规模集成电路）设计、计算机辅助设计以及统计学等领域中，都有计算几何的应用 [19]。关于这方面的研究，一本著名的专著是荷兰的 Berg 编著的《计算几何的算法与应用》[27]（中译本）。

### 2.4.2 在线算法

实际中很多问题都是在线（on-line）问题，即需要实时地进行调度，比如负载均衡、操作系统资源调度中的页面调度算法、实时路径规划、在线装箱等等。在线场景下，问题的所有信息无法事先知道，须要实时地根据已有

信息做出决策，因此往往不容易获得最优解。适用于在线场景下的算法就是在线算法。一般来说，在线算法也属于近似算法的一种，但与我们之前讨论的近似算法却是完全不同的概念。类似于对近似算法的分析，在线算法的性能通常用**竞争比**（Competitive Ratio）来衡量。对给定的一个极小化在线问题的一个实例 $I$，记某个在线算法的输出是 $ALG(I)$，对应的离线问题的最优解是 $OPT(I)$，如果对任意一个实例 $I$，都有 $ALG(I) \leq c \cdot OPT(I) + \alpha$（其中 $\alpha$ 是与输入实例无关的常量），我们就说在线算法 $ALG$ 是 $c$-竞争的。

除了一些基本的算法设计思想比如贪婪策略常被用于在线算法的设计外，近年来，许多数学优化技术、排队论、稳定性理论等也开始被广泛使用。比如，Niv Buchbinder 于 2008 年完成的博士论文 Designing Competitive Online Algorithms via a Primal-Dual Approach[1] 就详细讨论了原始——对偶方法在设计在线算法中的应用。而今年来随着智能终端、移动通信、云计算和大数据的迅速发展，云服务器、无线传感网络的在线调度算法设计成为研究的热点，在文献 [24, 23, 9, 15, 4] 中，就使用了排队论结合 Lyapunov 优化技术来解决服务器资源调度中的与能耗、费用、负载均衡等有关的一系列优化问题。

### 2.4.3 随机算法

随机算法在组合优化问题的求解中有着强大的生命力，现代工业级的优化设计中，一些智能的随机算法（如进化算法、蚁群系统、模拟退火等）展示出了强大的性能，开始发挥越来越重要的作用。随机算法是在算法执行决策的过程中引入随机扰动，从而有一定的概率跳出局部最优，搜索到更好的邻域。一个经典的随机算法是蒙特卡洛方法。随机算法的优点是往往可以克服“维数灾难”，且计算复杂性不大，缺点则是由于引入了随机扰动，会导致结果不稳定，而且一般都需要人为地给定一个初始解（对某些问题，初始解的获得并不容易），且算法运行的结果受初始解的影响比较大。随着现代智能优化算法的发展，在解决实际问题中，人们往往将随机算法与之前所提的确定算法融合起来，设计混合算法。比如，将确定性近似算法的解作为随机算法的初始解，这样随机算法的最坏情况就有了保证；又如，将近似算法作为随机算法的一个算子，所以现代的随机算法越来越复杂，也越来越智能。Marco Dorigo 这篇目前已经有七千多次引用量的论文 Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem[5] 就讨论了蚁群算法在求解 TSP 问题中发挥的作用，并阐述了将邻域搜索算法

（2-opt 和 3-opt）引入蚁群算法所带来的求解性能质的提升。Marco Dorigo 的在他的著作 Ant Colony Optimization 中也强调了蚁群算法和邻域搜索技术相结合的强劲性能 [6]。

### 2.4.4 近似算法的分类

正如前面所介绍的，除了可以从算法设计思想（贪婪、局部搜索等）的角度对近似算法进行分类，还可以从算法的近似性能的角度做分类。本节是对文献 [26] 的 15.6 内容的提炼。

**与最优值仅一步之遥的近似算法** 对于一些 NP-完全问题，人们已经找到一种极好的近似算法，它所得的近似值与原问题的精确最优解仅相差一个单位。两个经典的例子是简单图的边着色问题和最小——最大度支撑树问题。但是目前这样的问题并不多。然而为什么这些问题的近似解可以如此易于计算，以及如何识别一个问题是否存在这样的近似算法却是不容易回答的问题。

**定长近似比算法** 这是比较常见的近似算法，即近似比为常数的算法，我们称这种算法是“好的算法”，前面的章节中，我们介绍的近似算法基本都属于这一类。

**近似策略**（也叫**近似方案**）是指其近似比可以按所需任意地接近 1 的一类算法。一般来说，这类算法比定长近似比算法要好一些。但是往往在使近似比接近 1 的过程中，这类算法的时间复杂度将迅速增长，所以对于实际需求，须要在近似比和运行时间之间做一个权衡。但是如果一个问题存在**多项式时间近似策略**，基本就不用太顾虑于这种权衡。目前，人们已经找到了许多 NP-完全问题的多项式时间近似策略，比如背包问题（knapsack problem）、最小制造时跨问题（minimum makespan）等。

**最好可能近似比算法** 对某一特定问题，对任何的 $\epsilon > 0$，如果得到近似比为 $\delta - \epsilon$ 的近似算法是 NP-完全的，而近似比为 $\delta$ 的算法复杂度却是多项式时间的，称这种特性的近似算法是该问题的“最好的算法”。目前为止，所有具有上述特性的最好可能的近似算法仅仅对所谓的瓶颈类问题（bottleneck problems）得到实现（瓶颈类问题是包括各类通信网络设计问题、车辆路径问题等具有广泛应用领域的一类强 NP-完全问题）。

**比最好还要好的近似算法** 装箱问题是强 NP-完全的，除非 $P = NP$，它不可能存在多项式时间近似策略。Karp 在 1982 年给出了求解装箱问题的一个策略，其时间复杂度以物品的数目 $n$ 和 $\frac{1}{\epsilon}$ 的一个多项式函数为界，

并对任一实例 $I$ 给出解值不超过 $(1+\epsilon)OPT(I)+O((\frac{1}{\epsilon})^6)$ 的解。虽然这样一个策略是多项式时间近似策略，但其达到误差率 $\epsilon$ 是渐进意义上的。这个结果给我们的启示是，如果采用了不同的衡量近似算法性能的标准（如不采用绝对性能比而采用渐进性能比），对某些非常困难的问题，仍有可能导出性态不错的近似算法。

# 3 结论

近似算法的设计哲学是什么？它其实和“普通的算法”并没有什么两样，只是在设计近似算法的过程中，我们是有目标地在不大规模增加时间复杂度的前提下，尽可能地向最优解靠拢。而事实上，由于无法知晓真正的最优解形态，我们“靠拢”的，其实是最优解的一个界（下界或者上界）。所以很多时候，近似算法的设计基于对问题最优解“界”的掌控以及用何种方式导出这样的界，而这又依赖于我们对问题本身的认识以及基本算法的知识储备。

# 参考文献

[1] Niv Buchbinder and Thesis Seminar. Designing competitive online algorithms via a primal-dual approach. *Phd. thesis*, 2008.

[2] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley, New York, 1997.

[3] G. A. Croes. A method for solving traveling salesman problems. *Operations Research*, 6(6):791–812, 1958.

[4] Wei Deng, Fangming Liu, Hai Jin, and Chuan Wu. Smartdpss: Cost-minimizing multi-source power supply for datacenters with arbitrary demand. In *IEEE International Conference on Distributed Computing Systems*, pages 420–429, 2013.

[5] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[6] Marco Dorigo and Mauro Birattari. *Ant Colony Optimization*. The MIT Press, Cambridge, Massachusetts, 2004.

[7] Keld Helsgaun. Lkh. `http://www.akira.ruc.dk/~keld/research/LKH/`. Accessed Aug 10, 2016.

[8] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[9] L. Huang. Optimal sleep-wake scheduling for energy harvesting smart mobile devices. In *International Symposium on Modeling & Optimization in Mobile, Ad Hoc & Wireless Networks*, pages 484 – 491, 2013.

[10] Huawei Inc. 2016 华为软件精英挑战赛. `http://codecraft.huawei.com/`. Accessed Aug 10, 2016.

[11] D. S. Johnson and L. A. Mcgeoch. *The Traveling Salesman Problem: A Case Study*. Local Search in Combinatorial Optimization, 2010.

[12] D. S. Johnson, L. A. Mcgeoch, and E. E. Rothberg. Asymptotic experimental analysis for the held-karp traveling salesman bound. In *Acm-Siam Symposium on Discrete Algorithms*, pages 341–350, 1996.

[13] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[14] Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[15] Fangming Liu, Zhi Zhou, Hai Jin, Bo Li, Baochun Li, and Hongbo Jiang. On arbitrating the power-performance tradeoff in saas clouds. *Proceedings - IEEE INFOCOM*, 25(10):872–880, 2013.

[16] R C Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.

[17] Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis Ii. An analysis of several heuristics for thetraveling salesman problem. *Siam Journal on Computing*, 6(3):563–581, 1977.

[18] A. Schrijver. *Combinatorial optimization*. Springer, New York, 2003.

[19] Ronald L. Rivest Clifford Stein Thomas H. Cormen, Charles E. Leiserson. *Introduction to Algorithms*. MIT, 3ed. edition, 2009.

[20] V. V. Vazirani. *Approximation Algorithms*. Springer, second edition edition, 2003.

[21] D. B. West. *Introduction to Graph Theory*. Prenctice Hall, Upper Saddle River, 2nd edition edition, 2001.

[22] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[23] M. M. Zavlanos and G. J. Pappas. Distributed connectivity control of mobile networks. *IEEE Transactions on Robotics*, 24(6):1416–1428, 2008.

[24] Zhi Zhou, Fangming Liu, Yong Xu, Ruolan Zou, Hong Xu, John C. S. Lui, and Hai Jin. Carbon-aware load balancing for geo-distributed cloud services. In *IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 232–241, 2013.

[25] 堵丁柱. 近似算法的设计与分析. `http://mooc.chaoxing.com/course/55056.html#courseUnit`. Accessed Aug 10, 2016.

[26] 陈志平, 徐宗本. 计算机数学. 科学出版社, 2001.

[27] (荷) 伯格 (Berg and M.). 计算几何的算法与应用*:* 第 *3* 版. 世界图书出版公司北京公司, 2013.