

旅行商问题中巡回路径的数据结构

宗德才¹, 王康康²

(1. 常熟理工学院 计算机科学与工程学院, 江苏 常熟 215500;
2. 江苏科技大学 数理学院, 江苏 镇江 212003)

摘要:旅行商问题中巡回路径的数据结构对局部启发式算法的效率起着非常关键的作用。巡回路径的数据结构必须能够查询一条回路中每个城市的相对顺序,并且能够将一条回路中的部分城市逆序。分析了数组表示法、伸展树表示法和两级树表示法表示巡回路径时各种基本操作的实现过程及时间复杂度。数组表示法能够在常数时间内确定一条回路中每个城市的相对顺序,但是最坏情况下完成逆序操作需要 $O(n)$ 时间,不适用于大规模的旅行商问题。伸展树表示法执行查询和更新操作的平摊时间复杂度是 $O(\log n)$,适用于极大规模的旅行商问题。而两级树表示法在最坏情况下每一个更新操作的时间复杂度是 $O(n^{0.5})$,适用于大规模的旅行商问题。

关键词:旅行商问题;局部启发式算法;数组;伸展树;两级树

中图分类号:TP311.12

文献标识码:A

文章编号:1673-629X(2014)12-0072-06

doi:10.3969/j.issn.1673-629X.2014.12.018

Data Structure of Tour for Traveling Salesman Problem

ZONG De-cai¹, WANG Kang-kang²

(1. College of Computer Science and Engineering, Changshu Institute of Technology,
Changshu 215500, China;
2. School of Mathematics and Physics, Jiangsu University of Science and Technology,
Zhenjiang 212003, China)

Abstract: The data structure of tour for the traveling salesman problem plays a critical role in the efficiency of local heuristic algorithms. The data structure of tour must permit queries about the relative order of cities in the tour and must allow sections of the tour to be reversed. The implementation and time complexity of several basic operations when the tour is represented by array, splay tree and two-level tree are analyzed. The array-based representation of a tour permits the relative order of cities to be determined in small constant time, but requires worst-case $O(n)$ time to implement a reversal, which renders it impractical for large instances. For the data structure based on splay tree, all queries and updates take amortized time $O(\log n)$, which is available for extremely large-scale traveling salesman problem. For the data structure based on two-level tree representation, the worst-case time for each update operation is $O(n^{0.5})$, which is available for large-scale traveling salesman problem.

Key words: traveling salesman problem; local heuristic algorithm; array; splay tree; two-level tree

0 引言

旅行商问题(Traveling Salesman Problem, TSP)是一个典型的NP难^[1]的组合优化问题。

假设有 n 个城市,用 $1 \sim n$ 对城市进行编号, 1 表示城市 1 , 2 表示城市 2 , \dots , n 表示城市 n 。 Θ 是所有 $1, 2, \dots, n$ 排列的集合, $d_{i,j}$ 是城市 i 到城市 j 的距离, $s = (a_1, a_2, \dots, a_n)$ 是 $1, 2, \dots, n$ 的一个排列, $(a_1, a_2, \dots,$

$a_n)$ 表示访问的第一个城市是城市 a_1 ,第二个城市是城市 a_2, \dots ,访问的最后一个城市是 a_n ,最后再回到第一个城市 a_1 的一条巡回路径。当 $d_{i,j} = d_{j,i}$ 时,称为对称TSP问题;当 $d_{i,j} \neq d_{j,i}$ 时,称为不对称TSP问题。文中提到的TSP问题均是指对称TSP问题。

TSP问题可以用数学表达式表示为:

$$\min_{s \in \Theta} (d_{a_1, a_2} + d_{a_2, a_3} + \dots + d_{a_n, a_1}) \quad (1)$$

收稿日期:2014-01-24

修回日期:2014-04-28

网络出版时间:2014-10-23

基金项目:江苏省高校自然科学基金基础研究项目(13KJB110006);常熟理工学院青年教师基金项目(CST201209)

作者简介:宗德才(1979-),男,江苏常熟人,讲师,硕士,研究方向为智能优化算法、概率论;王康康,副教授,博士,研究方向为智能优化算法、概率论。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20141023.1102.023.html>

求解 TSP 问题的算法有两类:一类是精确算法,能够求出 TSP 问题的精确解,但是计算时间是问题规模(城市个数)的指数级,随着问题规模的扩大,将产生组合爆炸;另一类是随机优化算法或局部搜索算法,能够在多项式时间内求得问题的近似解。

近年来,为解决 TSP 问题出现了许多局部搜索算法,比较著名的有:2-opt 算法^[2]、3-opt 算法^[3]、Lin-Kernighan (LK)算法^[4]等。这些算法能够在较短时间内获得近似解。

在 2-opt 算法、3-opt 算法和 Lin-Kernighan 算法中,巡回路径的数据结构必须支持四种基本操作:Next(a)返回当前巡回路径中位于城市 a 之后的城市;Prev(a)返回当前巡回路径中位于城市 a 之前的城市;Between(a, b, c)返回 True 或 False,如果从城市 a 出发,先到达城市 b 再到达城市 c ,则返回 True,否则,返回 False;Flip(a, b, c, d)用边(b, c)和边(a, d)代替边(a, b)和边(c, d),假定 $a = \text{Next}(b)$, $d = \text{Next}(c)$ 。

1 巡回路径的数据结构

旅行商问题中的巡回路径可以用数组、伸展树和两级树来表示。下面将详细分析这三种表示方法的具体实现过程。

1.1 数组表示法

数组表示法是最简单的表示方法。文献[5-10]中巡回路径的数据结构都是用数组表示的。

一条巡回路径可以用两个长度是 n 的一维数组 A 和 B 表示。 A 表示巡回路径中访问城市的先后顺序, $A[1]$ 表示访问的第一个城市是 $A[1]$, $A[2]$ 表示访问的第二个城市是 $A[2]$, \dots , $A[n]$ 表示访问的最后一个城市是 $A[n]$; $B[1]$ 表示城市 c_1 在巡回路径中是第 $B[1]$ 个被访问的, $B[2]$ 表示城市 c_2 在巡回路径中是第 $B[2]$ 个被访问的, \dots , $B[n]$ 表示城市 c_n 在巡回路径中是第 $B[n]$ 个被访问的。

$$A[B[i]] = c_i, 1 \leq i \leq n$$

用数组表示法 Next(a), Prev(a), Between(a, b, c)操作能够在常数时间内完成。Next(a)操作的实现过程:假设 $a = c_k$,在数组 B 中找到 c_k 在回路中是第 $B[k]$ 个被访问的,Next(a) = $A[B[k] + 1]$ 。Prev(a)操作的实现过程:假设 $a = c_k$,在数组 B 中找到 c_k 在回路中是第 $B[k]$ 个被访问的,Prev(a) = $A[B[k] - 1]$ 。Between(a, b, c)操作的实现过程:假设 $a = c_i$, $b = c_j$, $c = c_k$,如果 $B[i] < B[j] < B[k]$,则 Between(a, b, c)返回 True,否则返回 False。

Flip(a, b, c, d)操作的实现最复杂。假设 $a = c_i$, $b = c_j$, $c = c_p$, $d = c_q$,令 $L_{ac} = B[p] - B[i] + 1$,如果 $L_{ac} < 0$, $L_{ac} = L_{ac} + n$,如果 $2L_{ac} < n$,则将 a 和 c 之间的路径反转,即

交换 $A[i]$ 与 $A[p]$ 的值, $A[i+1]$ 与 $A[p-1]$ 的值, \dots , 并且要交换 $B[i]$ 与 $B[p]$ 的值, $B[i+1]$ 与 $B[p-1]$ 的值, \dots ; 如果 $2L_{ac} > n$,则将 b 和 d 之间的路径反转。Flip(a, b, c, d)操作在最坏情况下的时间复杂度是 $O(n)$ 。

随着 n 的增加,这些操作所用的时间将占整个算法运行时间的绝大部分。

1.2 伸展树表示法

伸展树是一种能够实现自我调整的二叉查找树^[11],每当一个节点被访问后,它会沿着从该节点到树根的路径,通过一系列的旋转把该节点调整到树根处(伸展操作),使下次再访问该节点时可以快速找到该节点。文献[12]中巡回路径的数据结构是用伸展树表示的。

如果要对一个二叉查找树执行一系列操作,它能够在当前操作中使用一种简单的重建启发式规则(伸展)提高后续操作的效率。

一条巡回路径可以用一棵伸展树来表示,树中的每个节点代表一个城市,每个节点有一个反转位^[13](如果反转位是 0,表示将按照中序访问此节点的子树,即先访问该节点的左子树,然后访问该节点,最后访问右子树;如果反转位是 1,则先访问该节点的右子树,然后访问该节点,最后访问左子树)。如果节点的反转位是 1,则该节点用双圈表示。

由一棵带反转位的伸展树确定一条巡回路径的方法如下:从树根开始,从上往下将所有节点的反转位变成 0 得到一棵等价的树,然后中序遍历该树就可得到所需的巡回路径。如果一个节点的反转位是 1,通过以下方法可将该节点的反转位变为 0:首先交换该节点的左子树和右子树,并且给左子树和右子树的树根的反转位加上 1,如果反转位变成了 2,相当于取消反转,直接将反转位由 2 变成 0 即可。

对节点 x 执行伸展操作,是在保持伸展树有序性的前提下,通过一系列的伸展步骤将节点 x 调整到树根处。执行伸展操作之前要将相关节点的反转位置 0。

伸展步骤^[11]分三种:左旋(右旋)、右旋-右旋(左旋-左旋)、左旋-右旋(右旋-左旋)。

如图 1(a)所示,节点 x 的父节点 y 是根节点并且 x 是 y 的左孩子,则向右旋转连接节点 x 和节点 y 的边;如果 x 是 y 的右孩子,则向左旋转连接节点 x 和节点 y 的边。

如图 1(b)所示,节点 x 的父节点 y 不是根节点,节点 y 的父节点是 z ,并且 x 和 y 同时是各自父节点的左孩子,则先向右旋转连接节点 y 和 z 的边,然后向右旋转连接节点 x 和节点 y 的边;如果节点 x 的父节点 y 不是根节点,节点 y 的父节点是 z ,并且 x 和 y 同时是各自父节点的右孩子,则先向左旋转连接节点 y 和节

点 z 的边,然后向左旋转连接节点 x 和节点 y 的边。

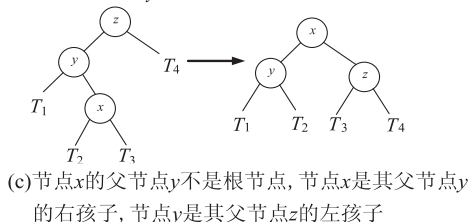
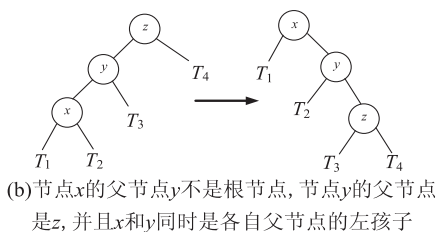
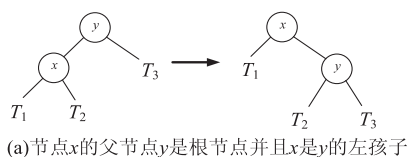


图 1 伸展步骤

如图 1(c) 所示, 节点 x 的父节点 y 不是根节点, 节点 x 是其父节点 y 的右孩子, 节点 y 是其父节点 z 的左孩子, 则先向左旋转连接节点 y 和节点 x 的边, 然后向右旋转连接节点 x 和节点 z 的边; 节点 x 的父节点 y 不是根节点, 节点 x 是其父节点 y 的左孩子, 节点 y 是其父节点 z 的右孩子, 则先向右旋转连接节点 y 和节点 x 的边, 然后向左旋转连接节点 x 和节点 z 的边。

伸展树的存储结构: 伸展树中一个节点对应一个城市, 可以使用散列法将伸展树的节点信息存储在一个数组 SPA 中, 城市 c_i 节点的信息存储在 SPA[1] 中, ..., 城市 c_i 节点的信息存储在 SPA[i] 中, 因此, 根据城市名字 c_i 可以在常数时间内找到城市 c_i 在伸展树中对应的节点位置。

Next(a) 操作的实现过程: 首先找到城市 a 在伸展树中的位置, 将相关节点的反转位变成 0, 通过伸展操作把该节点调整到树根处; 然后从树根开始, 将相关节点的反转位变成 0 后右子树中最左的节点就是 a 的后继节点 Next(a); 最后将 a 的后继节点 Next(a) 通过伸展操作调整到树根处。

Prev(a) 操作的实现过程: 首先找到城市 a 在伸展树中的位置, 将相关节点的反转位变成 0, 通过伸展操作把该节点调整到树根处; 然后从树根开始, 将相关节点的反转位变成 0 后左子树中最右的节点就是 a 的前驱节点 Prev(a); 最后将 a 的前驱节点 Prev(a) 通过伸展操作调整到树根处。

Between(a, b, c) 操作的实现过程: 首先找到城市 b 的位置, 通过伸展操作把节点 b 调整到树根处, 再找到节点 a 的位置, 通过伸展操作把节点 a 调整到树根处, 然后找到节点 c 的位置, 通过伸展操作把节点 c 调整

到树根处。找到节点 b 在伸展树中的新位置, 从节点 b 开始朝树根方向按照中序遍历伸展树, 如果节点 c 比节点 a 先找到并且节点 b 在节点 c 的左子树中或者节点 a 比节点 c 先找到并且节点 b 在节点 a 的右子树中, 则 Between(a, b, c) 返回 True, 否则返回 False。

Flip(a, b, c, d) 操作的实现最复杂。 首先找到节点 d 的位置, 通过伸展操作把节点 d 调整到树根处, 再找到节点 b 的位置, 通过伸展操作把节点 b 调整到树根处, 此时伸展树将处于下面四种情况之一, 如图 2 所示。图 2 中 $T_1 \sim T_4$ 是节点的子树。

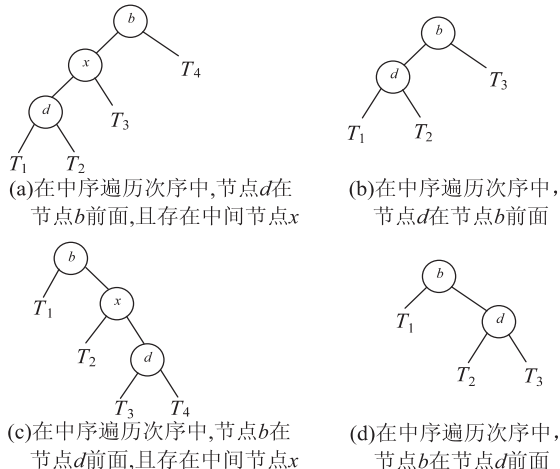


图 2 节点 d 和节点 b 先后进行伸展操作后伸展树的四种情况

对于图 2 中的 (a)、(b) 两种情况, Flip(a, b, c, d) 操作将对节点 b 和节点 d 之间的路径进行反转, 反转后的伸展树如图 3(a)、(b) 所示。对于图 2 中的 (c)、(d) 两种情况, Flip(a, b, c, d) 操作将对节点 a 和节点 c 之间的路径进行反转, 反转后的伸展树如图 3(c)、(d) 所示。图 3 中, 带双圈的节点的反转位是 1, T_i^k 表示子树 T_i 的根节点的反转位是 1。

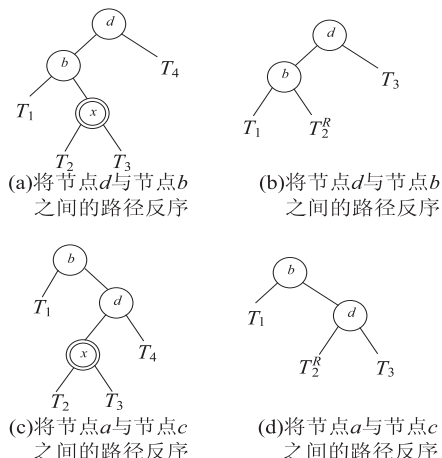


图 3 执行 Flip(a, b, c, d) 操作后的伸展树

对 n 个节点的伸展树表示的一条巡回路径执行一系列的 Next、Prev、Between、Flip 操作, 每个操作的平摊时间为 $O(\log n)^{[11]}$ (n 为回路中包含的城市个数)。

平摊时间是指执行一系列最坏情况下的每个操作的平均时间。

1.3 两级树表示法

在文献[14-15]中巡回路径的数据结构都是用两级树表示的。

一条巡回路径大约被分成 \sqrt{n} 段子路径,每段子路径中包含 $\sqrt{n}/2$ 到 $2\sqrt{n}$ 个城市。每段子路径用一个段节点表示,如图4所示。每个段节点有4个指针:prev、next分别指向前一段子路径和下一段子路径,first、last分别指向子路径中包含的第一个城市节点和最后一个城市节点,每段子路径中还有3个变量pos、rev、lens分别表示该段子路径在第一层链表中的位置,访问子路径中城市时是按顺时针方向还是按逆时针方向和该段子路径中包含的城市节点个数有关。每段子路径中的城市节点用双向链表相互连接起来。每个城市节点有3个指针:prevc(指向前驱节点)、nextc(指向后继节点)、parent(指向本节点所在的子路径),还有2个变量num、posc分别表示该城市的编号和在子路径中所处的位置。

图4中,两级树表示的一条巡回路径是($c_6, c_8, c_4, c_2, c_9, c_7, c_3, c_5, c_1$)。

两级树的存储结构:两级树中一个节点对应一个城市,可以使用散列法将两级树的节点信息存储在—

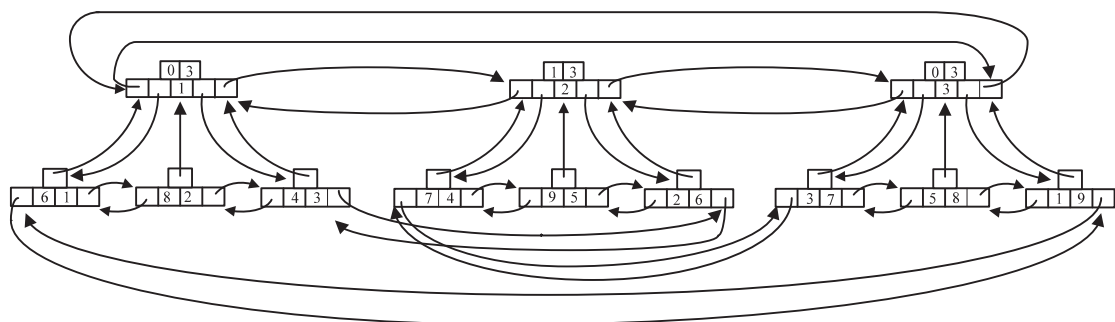
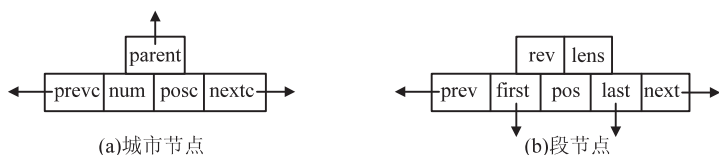
个数组 TLA 中,城市 c_i 节点的信息存储在 TLA[1] 中, ..., 城市 c_i 节点的信息存储在 TLA[i] 中,因此,根据城市名字 c_i 可以在常数时间内找到城市 c_i 在两级树中对应的节点位置。

由于在访问旅行回路时可以按照顺时针方向或逆时针方向访问,用一个变量 Reversed 表示在访问旅行回路时的方向,Reversed=0 表示顺时针方向,Reversed=1 表示逆时针方向。

Next(a)操作的实现过程:首先找到城市 a 在两级树中的节点位置 na,则城市节点 na 的父节点是 na->parent,如果 Reversed=0 并且 na->parent 段节点的 rev 值是 1,即按照逆时针方向访问 na->parent 段中的城市,则 Next(a) = a->prevc;如果 Reversed=0 并且 na->parent 段节点的 rev 值是 0,则 Next(a) = a->nextc;如果 Reversed=1 并且 na->parent 段节点的 rev 值是 1,则 Next(a) = a->nextc;如果 Reversed=1 并且 na->parent 段节点的 rev 值是 0,则 Next(a) = a->prevc。

Prev(a)操作的实现过程与 Next(a)操作的实现过程类似。

Between(a, b, c)操作的实现过程:首先找到城市 a, b, c 在两级树中的节点位置 na、nb、nc,则城市节点 a, b, c 的父节点分别是 na->parent、nb->parent、nc->parent。



(c)两级树表示的一条巡回路径

图4 两级树表示巡回路径

如果 na、nb、nc 在同一个段 na->parent 中,当 Reversed 等于 na->parent->rev 时,如果满足条件 na->posc < nb->posc < nc->posc 或者 nc->posc < na->posc < nb->posc 或者 nb->posc < nc->posc < na->posc,则 Between(a, b, c) 返回 True,否则返回 False;当 Reversed 不等于 na->parent->rev 时,如果满足条件 nb->posc < na->posc < nc->posc 或者 na->posc < nc->posc < nb->posc 或者 nc->posc < nb->posc < na->posc,则 Between(a, b, c) 返回 True,否则返回 False。

如果 na、nb、nc 有两个节点在同一个段中,如果 na、nb 在同一个段 na->parent 中,当 Reversed 等于 na->parent->rev 时,如果满足条件 na->posc < nb->posc,则 Between(a, b, c) 返回 True,否则返回 False;当 Reversed 不等于 na->parent->rev 时,如果满足条件 na->posc > nb->posc,则 Between(a, b, c) 返回 True,否则返回 False。如果 na、nc 在同一个段 na->parent 中,当 Reversed 等于 na->parent->rev 时,如果满足条件 nc->posc < na->posc,则 Between(a, b, c) 返回 True,否则返

回 False; 当 Reversed 不等于 $na \rightarrow parent \rightarrow rev$ 时, 如果满足条件 $na \rightarrow posc < nc \rightarrow posc$, 则 $Between(a, b, c)$ 返回 True, 否则返回 False。如果 nb, nc 在同一个段 $nb \rightarrow parent$ 中, 当 Reversed 等于 $nb \rightarrow parent \rightarrow rev$ 时, 如果满足条件 $nb \rightarrow posc < nc \rightarrow posc$, 则 $Between(a, b, c)$ 返回 True, 否则返回 False; 当 Reversed 不等于 $nb \rightarrow parent \rightarrow rev$ 时, 如果满足条件 $nc \rightarrow posc < nb \rightarrow posc$, 则 $Between(a, b, c)$ 返回 True, 否则返回 False。

如果 na, nb, nc 在三个不同的段中, 当 Reversed=0 时, 如果满足条件 $na \rightarrow parent \rightarrow pos < nb \rightarrow parent \rightarrow pos < nc \rightarrow parent \rightarrow pos$ 或者 $nc \rightarrow parent \rightarrow pos < na \rightarrow parent \rightarrow pos < nb \rightarrow parent \rightarrow pos$ 或者 $nb \rightarrow parent \rightarrow pos < nc \rightarrow parent \rightarrow pos < na \rightarrow parent \rightarrow pos$, 则 $Between(a, b, c)$ 返回 True, 否则返回 False; 当 Reversed=1 时, 如果满足条件 $nb \rightarrow parent \rightarrow pos < na \rightarrow parent \rightarrow pos < nc \rightarrow parent \rightarrow pos$ 或者 $na \rightarrow parent \rightarrow pos < nc \rightarrow parent \rightarrow pos < nb \rightarrow parent \rightarrow pos$ 或者 $nc \rightarrow parent \rightarrow pos < nb \rightarrow parent \rightarrow pos < na \rightarrow parent \rightarrow pos$, 则 $Between(a, b, c)$ 返回 True, 否则返回 False。

Flip(a, b, c, d) 操作的实现最复杂。

如果节点 a 和节点 c 之间的路径在同一个段中并且节点 a 和节点 c 之间的长度大于 $3/4\sqrt{n}$ (如图 5(a) 所示), 那么, 当节点 a 和节点 b 在同一段中时, 将该段在节点 a 和节点 b 中间分成两段并且将长度较短的段与邻接段合并以保持段的总个数不变, 当节点 c 和节点 d 在同一段中时, 将该段在节点 c 和节点 d 中间分成两段并且将长度较短的段与邻接段合并以保持段的总个数不变, 然后在同一个段中将节点 a 和节点 c 之间的路径反转, 即更新节点 a 和节点 c 之间的城市节点的 nextc、prevc 指针和 posc 变量的值以及节点 b, d 的 nextc、prevc 指针的值。

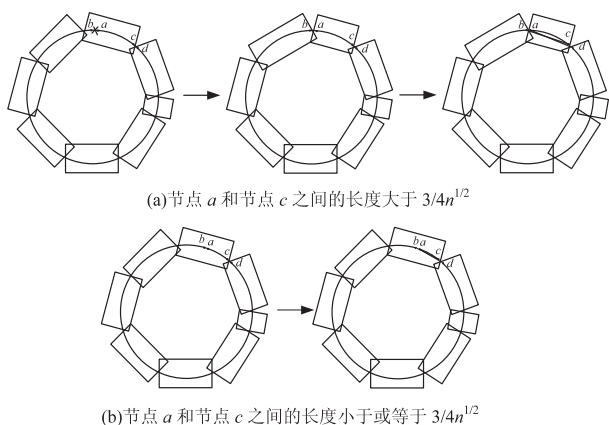


图5 节点 a 和节点 c 之间的路径在同一个段中

如果节点 a 和节点 c 之间的路径在同一个段中并且节点 a 和节点 c 之间的长度小于或等于 $3/4\sqrt{n}$ (如图 5(b) 所示), 那么, 在同一个段中将节点 a 和节点 c

之间的路径反转, 即更新节点 a 和节点 c 之间的城市节点的 nextc、prevc 指针和 posc 变量的值以及节点 b, d 的 nextc、prevc 指针的值。如果节点 b 和节点 d 之间的路径在同一个段中, 处理方法类似。

如果节点 b 和节点 d 之间的路径与节点 a 和节点 c 之间的路径都是由多个连续的完整段组成的 (如图 6 所示), 假设段的总个数为 S , 令 $L_{ac} = c \rightarrow parent \rightarrow pos - c \rightarrow parent \rightarrow pos + 1$, 如果 $L_{ac} < 0$, $L_{ac} = L_{ac} + S$ 。如果 $2L_{ac} > S$, 则将节点 b 和节点 d 之间的路径反转, 即更新 b 和 d 之间的段节点的 prev、next 指针和 pos 变量的值以及 $a \rightarrow parent, c \rightarrow parent$ 段节点的 prev、next 指针的值。如果 $2L_{ac} < S$, 则将节点 a 和节点 c 之间的路径反转, 即更新 a 和 c 之间的段节点的 prev、next 指针和 pos 变量的值以及 $b \rightarrow parent, d \rightarrow parent$ 段节点的 prev、next 指针的值。

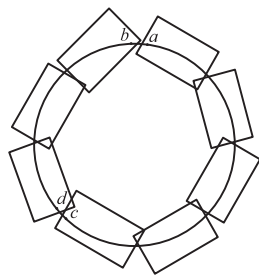


图6 节点 b 和节点 d 之间的路径与节点 a 和节点 c 之间的路径都是由多个连续的完整段组成

如果节点 b 和节点 d 之间的路径不在同一个段中并且节点 a 和节点 c 之间的路径也不在同一个段中, 那么, 当节点 a 和节点 b 在同一段中时, 将该段在节点 a 和节点 b 中间分成两段并且将长度较短的段与邻接段合并以保持段的总个数不变, 当节点 c 和节点 d 在同一段中时, 将该段在节点 c 和节点 d 中间分成两段并且将长度较短的段与邻接段合并以保持段的总个数不变, 如果分段后, 节点 b 和节点 d 之间的路径在同一个段中或节点 a 和节点 c 之间的路径在同一个段中, 则处理方法如图 5 所示。如果分段后, 节点 b 和节点 d 之间的路径不在同一个段中并且节点 a 和节点 c 之间的路径也不在同一个段中, 则处理方法如图 6 所示。

Prev(a)、Next(a) 和 Between(a, b, c) 操作都能够能够在常数时间内完成, 而 Flip(a, b, c, d) 操作最坏情况下的时间复杂度是 $O(\sqrt{n})$ [14]。

2 结束语

在 2-opt 算法、3-opt 算法和 Lin-Kernighan 算法等局部搜索算法中, 巡回路径的数据结构必须支持四种基本操作, 即 Next(a)、Prev(a)、Between(a, b, c) 和 Flip(a, b, c, d)。旅行商问题中的巡回路径可以用数组、伸展树和两级树来表示。文中主要详细分析了这

三种表示方法的具体实现过程以及每一种数据结构中四种基本操作的时间复杂度。当巡回路径用数组表示时, $\text{Next}(a)$ 、 $\text{Prev}(a)$ 和 $\text{Between}(a, b, c)$ 操作能够在常数时间内完成, $\text{Flip}(a, b, c, d)$ 操作在最坏情况下的时间复杂度是 $O(n)$, 不适用于大规模的旅行商问题。当巡回路径用伸展树表示时, 每个操作的平摊时间复杂度为 $O(\log n)$, 适用于极大规模的旅行商问题。当巡回路径用两级树表示时, $\text{Prev}(a)$ 、 $\text{Next}(a)$ 和 $\text{Between}(a, b, c)$ 操作都能够在常数时间内完成, 而 $\text{Flip}(a, b, c, d)$ 操作在最坏情况下的时间复杂度是 $O(n^{1/2})$, 适用于大规模的旅行商问题。下一步将用 C 语言编程实现这三种数据结构, 并通过仿真实验分析这三种数据结构对 Lin-Kernighan 算法等局部搜索算法在求解旅行商问题时性能的影响。

参考文献:

- [1] Garey M R, Johnson D S. Computers and intractability: a guide to the theory of NP-completeness [M]. San Francisco, USA: Freeman W H, 1979.
- [2] Croes G A. A method for solving traveling salesman problems [J]. Operations Research, 1958, 6(6): 791-812.
- [3] Lin S. Computer solutions of the traveling salesman problem [J]. Bell System Technical Journal, 1965, 44(10): 2245-2269.
- [4] Lin S, Kernighan B W. An effective heuristic algorithm for the traveling salesman problem [J]. Operations Research, 1973, 21(2): 498-516.
- [5] 宗德才, 王康康. 改进的嵌套分区算法求解旅行商问题 [J]. 计算机工程与应用, 2011, 47(24): 54-57.
- [6] 陈晓峰, 宋杰. 量子人工鱼群算法 [J]. 东北大学学报 (自然科学版), 2012, 33(12): 1710-1713.
- [7] 苏晓勤, 孙鹤旭, 潘旭华. 改进蜂群算法的旅行商问题仿真 [J]. 计算机工程与设计, 2013, 34(4): 1420-1424.
- [8] 王忠英, 白艳萍, 岳利霞. 经过改进的求解 TSP 问题的蚁群算法 [J]. 数学的实践与认识, 2012, 24(4): 133-140.
- [9] 柳寅, 马良. 模糊人工蜂群算法的旅行商问题求解 [J]. 计算机应用研究, 2013, 30(9): 2694-2696.
- [10] 申铨京, 刘阳阳, 黄永平, 等. 求解 TSP 问题的快速蚁群算法 [J]. 吉林大学学报 (工学版), 2013, 43(1): 147-151.
- [11] Sleator D D, Tarjan R E. Self-adjusting binary search trees [J]. Journal of the Association for Computing Machinery, 1985, 32(3): 652-686.
- [12] Gamboa D, Rego C, Glover F. Data structures and ejection chains for solving large-scale traveling salesman problems [J]. European Journal of Operational Research, 2005, 160(1): 154-171.
- [13] Chrobak M, Szymacha T, Krawczyk A. A data structure useful for finding Hamiltonian cycles [J]. Theoretical Computer Science, 1990, 71(3): 419-424.
- [14] Helsgaun K. An effective implementation of the Lin-Kernighan traveling salesman heuristic [J]. European Journal of Operational Research, 2000, 126(1): 106-130.
- [15] Helsgaun K. General k-opt submoves for the Lin-Kernighan TSP heuristic [J]. Mathematical Programming Computation, 2009, 1(2-3): 119-163.
- [1] Garey M R, Johnson D S. Computers and intractability: a guide to the theory of NP-completeness [M]. San Francisco, USA: Freeman W H, 1979.
- [2] Croes G A. A method for solving traveling salesman problems [J]. Operations Research, 1958, 6(6): 791-812.
- [3] Lin S. Computer solutions of the traveling salesman problem [J]. Bell System Technical Journal, 1965, 44(10): 2245-2269.
- [4] Lin S, Kernighan B W. An effective heuristic algorithm for the traveling salesman problem [J]. Operations Research, 1973, 21(2): 498-516.
- [5] 宗德才, 王康康. 改进的嵌套分区算法求解旅行商问题 [J]. 计算机工程与应用, 2011, 47(24): 54-57.
- [6] 陈晓峰, 宋杰. 量子人工鱼群算法 [J]. 东北大学学报 (自然科学版), 2012, 33(12): 1710-1713.
- [7] 苏晓勤, 孙鹤旭, 潘旭华. 改进蜂群算法的旅行商问题仿真 [J]. 计算机工程与设计, 2013, 34(4): 1420-1424.
- [8] 王忠英, 白艳萍, 岳利霞. 经过改进的求解 TSP 问题的蚁群算法 [J]. 数学的实践与认识, 2012, 24(4): 133-140.
- [9] 柳寅, 马良. 模糊人工蜂群算法的旅行商问题求解 [J]. 计算机应用研究, 2013, 30(9): 2694-2696.
- [10] 申铨京, 刘阳阳, 黄永平, 等. 求解 TSP 问题的快速蚁群算法 [J]. 吉林大学学报 (工学版), 2013, 43(1): 147-151.
- [11] Sleator D D, Tarjan R E. Self-adjusting binary search trees [J]. Journal of the Association for Computing Machinery, 1985, 32(3): 652-686.
- [12] Gamboa D, Rego C, Glover F. Data structures and ejection chains for solving large-scale traveling salesman problems [J]. European Journal of Operational Research, 2005, 160(1): 154-171.
- [13] Chrobak M, Szymacha T, Krawczyk A. A data structure useful for finding Hamiltonian cycles [J]. Theoretical Computer Science, 1990, 71(3): 419-424.
- [14] Helsgaun K. An effective implementation of the Lin-Kernighan traveling salesman heuristic [J]. European Journal of Operational Research, 2000, 126(1): 106-130.
- [15] Helsgaun K. General k-opt submoves for the Lin-Kernighan TSP heuristic [J]. Mathematical Programming Computation, 2009, 1(2-3): 119-163.

(上接第 71 页)

- [2] 刘洲洲. 基于 HSI 空间伪彩色异构多传感器图像融合仿真 [J]. 计算机技术与发展, 2013, 23(10): 201-203.
- [3] Ma Jianping, Jiang Jin. Detection and identification of faults in NPP instruments using kernel principal component analysis [J]. Journal of Engineering for Gas Turbines and Power, 2012, 134(3): 739-744.
- [4] Wang Rui, Wu Yi, Ding mingyue, et al. Medical image fusion based on spiking cortical model [J]. Proc of SPIE, 2013, 8676: 101-104.
- [5] 江铁, 朱桂斌, 孙奥. 基于金字塔变换的多曝光图像融合 [J]. 计算机技术与发展, 2013, 23(1): 95-98.
- [6] 王海江, 王周龙, 吴孟泉, 等. 一种最佳基小波包变换的影像融合研究 [J]. 测绘科学, 2009, 34(5): 63-66.
- [7] 蒋年德, 毛建旭, 王耀南. 基于小波包变换的遥感图像融合 [J]. 微计算机信息, 2008, 24(13): 286-287.
- [8] 李林宜, 李德仁. 基于最佳小波包变换的遥感影像融合方法 [J]. 计算机应用研究, 2007, 24(3): 318-320.
- [9] 赵松年, 熊小芸. 子波变换与子波分析 [M]. 北京: 电子工业出版社, 1996.
- [10] Coifman R R, Wickerhauser M V. Entropy-based algorithms for best basis selection [J]. IEEE Transactions on Information Theory, 1992, 38(2): 713-718.
- [11] Wickerhauser M V. Comparison of picture compression methods: wavelet, wavelet packet, and local cosine transform coding [C]//Proceedings of the international conference on wavelets: theory, algorithms, and applications. Taormina, Sicily: [s. n.], 1993: 585-621.
- [12] Garbas Jens-Uwe, Presquet-Popescu B, Trocan M, et al. Wavelet-based multi-view video coding with joint best basis wavelet packets [C]//Proceedings of ICIP. San Diego, CA: IEEE, 2008: 1232-1235.
- [13] 夏明革, 何友, 欧阳文. 像素级图像融合方法与融合效果评价 [J]. 遥感技术与应用, 2002, 17(4): 224-229.
- [14] 胡良梅, 高隽, 何柯峰. 图像融合质量评价方法的研究 [J]. 电子学报, 2004, 32(F12): 218-221.

旅行商问题中巡回路径的数据结构

作者：[宗德才](#)，[王康康](#)，[ZONG De-cai](#)，[WANG Kang-kang](#)
作者单位：[宗德才, ZONG De-cai \(常熟理工学院 计算机科学与工程学院, 江苏 常熟, 215500\)](#)，[王康康, WANG Kang-kang \(江苏科技大学 数理学院, 江苏 镇江, 212003\)](#)
刊名：[计算机技术与发展](#)[ISTIC](#)
英文刊名：[Computer Technology and Development](#)
年，卷(期)：2014 (12)

引用本文格式：[宗德才](#). [王康康](#). [ZONG De-cai](#). [WANG Kang-kang](#) [旅行商问题中巡回路径的数据结构](#) [期刊论文]-[计算机技术与发展](#) 2014 (12)