第 24 卷 第 9 期
2003 年 9 月

小 型 微 型 计 算 机 系 统
MINI-MICRO SYSTEMS

Vol. 24 No. 9
Sep. 2003

带权区间图的最短路算法

王晓东, 吴英杰

(福州大学 计算机科学与技术系, 福建 福州 350002)

摘　要: 提出一个解带权区间图的最短路问题的 $O(n\alpha(n))$ 时间新算法, 其中 n 是带权区间图中带权区间的个数, $\alpha(n)$ 是单变量 Ackerman 函数的逆函数, 它是一个增长速度比 $\log n$ 慢得多的函数, 对于通常所见到的 n, $\alpha(n) \leq 4$. 本文提出的新算法不仅在时间复杂性上比直接用 Dijkstra 算法解带权区间图的最短路问题有较大改进, 而且算法设计思想简单, 易于理解和实现.

关 键 词: 最短路; 区间图; 并查集

中图分类号: TP30　　**文献标识码**: A　　**文 章 编 号**: 1000-1220(2003)09-1655-03

A New Algorithm for Shortest Paths on Weighted Interval Graphs

WANG Xiao-dong, WU Ying-jie

(*Computer Science Department of Fuzhou University, Fuzhou 350002, China*)

Abstract: This paper presents a new $O(n\alpha(n))$ time algorithm for computing single source shortest paths in a weighted interval graph, where n is the number of weighted intervals. $\alpha(n)$ is the functional inverse of Ackermann's function. Its growing rate is much slower than that of function $\log n$. For commonly used n, $\alpha(n) \leq 4$. The new algorithm is not only an improvement in time complexity compared to the algorithm using Dijkstra algorithm directly, but also simple and easy to implement.

Key words: shortest paths; interval graphs; union find algorithms

1　引　言

S 是直线上 n 个带权区间的集合. 从区间 $I \in S$ 到区间 $J \in S$ 的一条路是 S 的一个区间序列 J(1), J(2), …, J(k), 其中 J(1) = I, J(k) = J, 且对所有 $1 \leq i \leq k-1$, J(i) 与 J(i+1) 相交. 这条路的长度定义为路上各区间权之和. 在所有从 I 到 J 的路中, 路长最短的路称为从 I 到 J 的最短路. 带权区间图的单源最短路问题要求计算从 S 中一个特定的源区间到 S 中所有其它区间之间的最短路[2,4].

定义 1: 一个区间 I 包含另一个区间 J 当且仅当 $I \cap J = J$.

定义 2: 区间 I 交区间 J 当且仅当 $I \cap J \neq \varnothing$.

设集合 S 由区间 I(1), I(2), …, I(n) 构成

$I(i) = [a(i), b(i)]$, $1 \leq i \leq n$; $b(1) \leq b(2) \leq \cdots \leq b(n)$.

区间 I(i) 带权 $w(i) > 0$, $1 \leq i \leq n$.

定义 3: 由区间 I(1), I(2), …, I(i) 构成的集合记为 S(i), $1 \leq i \leq n$.

不失一般性, 设区间 I(1), I(2), …, I(n) 的并覆盖了从 a(1) 到 b(n) 的线段. 源区间是 I(1).

对于任一区间集 S, 其并集可能有多个连通分量. 若区间 I 和 J 分别属于 S 的并集的 2 个不同的连通分量, 则区间 I 和 J 在 S 中没有路.

2　算法设计思想

2.1　基本算法

定义 4: 区间集 S(i) 的扩展定义为: $S(i) \cup T$, 其中 T 是满足下面条件的另一区间集. T 中任意区间 $I = [a, b]$ 均有 $b > b(i)$.

定义 5: 设区间 I(k) ($k \leq i$) 是区间集 S(i) 中的一个区间, $1 \leq i \leq n$. 如果对于 S(i) 的任意扩展 $S(i) \cup T$, 当区间 $J \in T$ 且在 $S(i) \cup T$ 中有从 I(1) 到 J 的路时, 在 $S(i) \cup T$ 中从 I(1) 到 J 的任一最短路都不含区间 I(k), 则称区间 I(k) 是 S(i) 中的无效区间. 若 S(i) 中的区间 I(k) 不是无效区间则称其为 S(i) 中的有效区间.

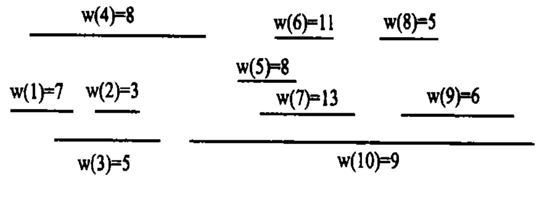

图 1　区间集

Fig. 1　An interval set

在图 1 中, 区间 I(2) 是 S(4) 中的无效区间; 区间 I(3) 是 S(4) 中的有效区间; 区间 I(5) 是 S(5) 中的无效区间; 区间 I(9)

收稿日期: 2002-03-21　基金项目: 国家自然科学基金项目(60172017)资助; 福建省科技厅杰出人才基金项目(2000Z148)资助.　作者简介: 王晓东, 教授, 研究领域为数据结构, 算法设计与分析. 吴英杰, 硕士研究生, 研究领域为数据结构, 算法设计与分析.

是 S(10) 中的无效区间; 区间 I(10) 是 S(10) 中的有效区间.

性质 1: 区间 I(k) 是 S(i) 中的有效区间, 则对任意 $k \le j \le i$, 区间 I(k) 是 S(j) 中的有效区间. 另一方面, 若区间 I(k) 是 S(i) 中的无效区间, 则对任意 $j > i$, 区间 I(k) 是 S(j) 中的无效区间.

性质 2: 集合 S(i) 中所有有效区间的并覆盖从 a(1) 到 b(j) 的线段, 其中 b(j) 是 S(i) 的最右有效区间的右端点.

证明: 对 S(i) 中任一有效区间 I(k), $k \le i$, 由定义可知, 在 S(i) 中有一条从 I(1) 到 I(k) 的最短路. 这意味着这条最短路上所有区间均为 S(i) 中有效区间. 由此可见, 若 b(j) 是 S(i) 的最右有效区间的右端点, 则集合 S(i) 中所有有效区间的并覆盖从 a(1) 到 b(j) 的线段.

性质 3: 区间 I(i) 是集合 S(i) 中的有效区间当且仅当在 S(i) 中有一条从 I(1) 到 I(i) 的路.

定义 6: S(j) 中从 I(1) 到 I(i) 的最短路长记为 dist(i, j), $i \le j$. 当 $i > j$ 时, dist(i, j) = $+\infty$.

由上面的定义可知, 对所有 i 均有, dist(i, 1) $\ge$ dist(i, 2) $\ge \cdots \ge$ dist(i, n).

若在 S(i) 中不存在从 I(1) 到 I(k) 的路, 则对 $k \le j \le i$ 有 dist(j, i) = $+\infty$.

在图 1 中, dist(8, 9) = $+\infty$, dist(8, 10) = 29.

性质 4: 当 $i > k$ 且 dist(i, i) < dist(k, i) 时, I(k) 是 S(i) 中的无效区间.

证明: 由 dist(i, i) < dist(k, i) 知, dist(i, i) < $+\infty$. 从而在 S(i) 中有一条从 I(1) 到 I(i) 的路和一条从 I(1) 到 I(k) 的路. 由 dist(i, i) < dist(k, i) 可推知, S(i) 中从 I(1) 到 I(i) 的最短路中不含区间 I(k). 由此可见, I(k) 是 S(i) 中的无效区间. ■

性质 5: 设 I(j(1)), I(j(2)), …, I(j(k)) 是 S(i) 中的有效区间, 且 $j(1) < j(2) < \cdots < j(k) \le i$, 则 dist(j(1), i) $\le$ dist(j(2), i) $\le \cdots \le$ dist(j(k), i).

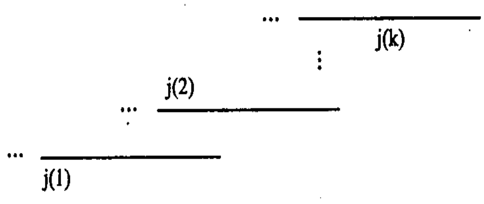

图 2 有效区间的单调性

Fig. 2 Monotonicity of the effective intervals

性质 6: 如果区间 I(i) 包含区间 I(k) (因此 $i > k$), 且 dist(i, i) < dist(k, i), 则 I(k) 是 S(i) 中的无效区间.

性质 7: 当 $i > k$ 且 dist(i, i) < dist(k, i-1) 时, I(k) 是 S(i) 中的无效区间.

证明: 由 dist(i, i) < dist(k, i-1) 知, dist(i, i) < $+\infty$. 从而在 S(i) 中有一条从 I(1) 到 I(i) 的路和一条从 I(1) 到 I(k) 的路. 分 2 种情况讨论.

(1) S(i) 中从 I(1) 到 I(k) 的最短路中不含区间 I(i). 此时, dist(k, i) = dist(k, i-1), 因此 dist(i, i) < dist(k, i). 由性质 6 即知 I(k) 是 S(i) 中的无效区间.

(2) S(i) 中从 I(1) 到 I(k) 的最短路中含区间 I(i). 因此, dist(k, i) $\ge$ dist(i, i) + w(k) > dist(i, i), (由于 w(k) > 0), 又由性质 6 知 I(k) 是 S(i) 中的无效区间. ■

性质 8: 如果区间 I(k) ($k > 1$) 不包含 S(k-1) 中任一有效区间 I(j) 的右端点 b(j), 则对任意 $i \ge k$, I(k) 是 S(i) 中的无效区间.

证明: 只要证明 I(k) 是 S(k) 中的无效区间就够了. 假设 I(k) 是 S(k) 中的有效区间. 由性质 2 知, S(k) 中所有有效区间的并覆盖从 a(1) 到 b(k) 的线段. 因此区间 I(k) 包含了 S(k) 中不同于 I(k) 的另一有效区间 I(j) 的右端点 b(j) ($j < k$). 由于 I(j) $\in$ S(k-1) = S(k) - {I(k)}, 由假设即知, I(j) 是 S(k-1) 中的无效区间, 从而 I(j) 也是 S(k) 中的无效区间. 此为矛盾. 因此, I(k) 是 S(k) 中的无效区间. ■

根据上面的讨论可设计带权区间图的最短路算法如下.

```
算法 Shortest Interval Paths
步骤 1: dist(1, 1) ←w(1);
步骤 2:
        for (i= 2; i< = n; i+ + ) {
            (2.1):
                j= min{ k | a(i)< b(k); 1≤k< i };
                if (j 不存在) dist(i, i)←+∞;
                else dist(i, i)←dist(j, i-1)+ w(i);
            (2.2):
                for (k< i){
                if (dist(i, i)< dist(k, i-1)) dist(k, i)←+∞;
                else dist(k, i)←dist(k, i-1);
                }
            }
步骤 3:
        for (i= 2; i< = n; i+ + ) {
            if (dist(i, n)= +∞) {
                j= min{ k | (dist(k, n)< +∞)&&(a(i)< b(k)) };
                dist(i, n)= dist(j, n)+ w(i);
                }
            }
```

上述算法的关键是有效地实现步骤 (2.1) 和 (2.2).

2.2 实现方案 1

用一棵平衡搜索树 (2-3 树) 来存储当前区间集 S(i) 中的

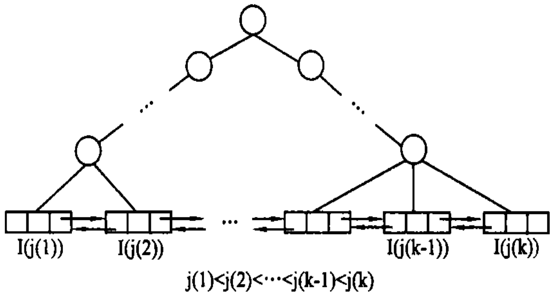

图 3 存储有效区间的平衡搜索树

Fig. 3 A balanced search tree for storing the effective intervals

有效区间. 以区间的右端点的值为序. 如图 3 所示.

(2.1) 的实现对应于平衡搜索树从根到叶的一条路径上的搜索, 在最坏情况下需要 $O(\log n)$ 时间.

(2.2) 的实现对应于反复检查并删除平衡搜索树中最右叶结点的前驱结点. 在最坏情况下, 每删除一个结点需要 $O(\log n)$ 时间.

综上, 算法 Shortest Interval Paths 用平衡搜索树的实现方案, 在最坏情况下的计算时间复杂性为 $O(n \log n)$.

2.3 实现方案 2

并查集结构[1,3,5]. 用整数 k 表示区间 I(k), 1≤k≤n. 初始时每个元素 k 构成一个单元素集, 即集合 k 是{k}, 1≤k≤n.

(1) 每个当前有效区间 I(k) 在集合 k 中. 设 I(i(1)), I(i(2)), …, I(i(k)) 是 S(i) 中的有效区间, 且 i(1) < i(2) < … < i(k). 对任意 1≤j≤k-1, 集合{h | i(j) < h < i(j+1)}包含在集合 i(j+1) 中. 依此定义, 集合 i(j+1) 中包含界于有效区间 I(i(j)) 和 I(i(j+1)) 之间的所有无效区间和有效区间 I(I(j+1)) 的区间序号.

(2) 对每个集合 S(i), 设

L(S(i)) = {I(k) | I(k) 是 S(i) 的无效区间, 且 I(k) 与 S(i) 的任一有效区间均不相交}

例如, 在图 1 中, S(9) 的有效区间是 I(1), I(3), I(4); L(S(9)) = {I(5), I(6), I(7), I(8), I(9)}.

由性质 2 可知, L(S(i)) 中所有区间均位于 S(i) 的所有有效区间并的右侧. L(S(i)) 非空当且仅当 I(i) ∈ L(S(i)). 当 L(S(i)) 非空时, L(S(i)) 中所有无效区间的序号存放在集合 i 中.

(3) 用一个栈 AS 来存放当前有效区间 I(i(1)), I(i(2)), …, I(i(k)). I(i(k)) 是栈顶元素. 该栈称为当前有效区间栈.

(4) 对于 1≤k≤n, 记 prev(I(k)) = min{j | a(k) < b(j)}.

例如, 在图 1 中, prev(I(5)) = 5, prev(I(9)) = 8, prev(I(10)) = 4.

prev(I(k)) ≤ k; 仅当区间 I(k) 不含其它区间的右端点时 prev(I(k)) = k.

由于 prev(I(k)) 的定义是静态的, 可以对给定的区间序列做一次线性扫描确定 prev(I(k)) 的值.

(5) 对于当前区间集 S(i), 用一维数组 dist 记录 dist(j, i) 的值.

(6) 用 dist[k] = -1 来标记区间 I(k) 为无效区间.

基于上述并查集结构, 基本算法中的步骤 2 可实现如下.

```
算法 Shortest Interval Paths
{
    步骤 1:
    UF. Init(n); // 初始化并查集 UF
    Preprocess(prev);
    // 计算 prev 的值使 prev[i] = min{ k | a[i] ≤ b[k]; 0≤k ≤ i }
    dist[0] = w[0];
    AS. PUSH(0);
    步骤 2:
    for (int i = 1; i < n; i++) {
      int j = UF. Find(prev[i]);
      (2.1):
      // I(i) 与 S(i-1) 中区间不交, 是 S(i) 中无效区间;
      // I(i) 与 S(i-1) 中有效区间不交, 是 S(i) 中无效区间;
      if ((j == i) || (j == i-1) && (dist[i-1] < 0)) {
        dist[i] = -1;
        if (dist[i-1] < 0) UF. Union(i-1, i);
      }
      (2.2):
      // I(j) 是 S(i-1) 中的有效区间;
      if ((j < i) && (dist[j] > 0)) {
        dist[i] = dist[j] + w[i];
        if (dist[i-1] < 0) UF. Union(i-1, i);
        // 对栈 AS 作如下调整:
        while (! AS. EMPTY()) {
          int k = AS. TOP();
          if (dist[i] < dist[k]) {
            AS. POP(k);
            dist[k] = -1;
            UF. Union(k, i);
          }
          else break;
        }
        AS. PUSH(i);
      }
    }
    步骤 3:
    Process(prev);
    // 计算 prev 的值使 prev[i] = min{ k | (dist[k] > 0) && (a[i] ≤ b[k]); 0≤k < n }
    for (int i = 1; i < n; i++)
      if (dist[i] < 0) dist[i] = dist[prev[i]] + w[i];
}
```

容易看到, 上述算法总共执行 O(n) 次 Union 和 Find 运算. 由此可见, 在最坏情况下, 算法需要 $O(n\alpha(n))$ 计算时间[5], 其中 $\alpha(n)$ 是单变量 Ackerman 函数的逆函数, 它是一个增长速度比 $\log n$ 慢得多的函数, 对于通常所见到的 n, $\alpha(n) \le 4$.

References

1 Aho A V, Hopcroft J E and Ullman J D. The Design and analysis of computer algorithms [M]. Addison Wesley, Reading, Massachusetts, 1974, 124～147.

2 Booth K S and Lueker G S. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ tree algorithms [J]. Journal of Computer and System Sciences, 1976, 13: 335～379.

3 Golumbic M C. Algorithmic graph theory and perfect graphs [M]. Academic Press, New York, 1980.

4 Gupta U I, Lee D T and Leung J Y T. Efficient algorithms for interval graphs and circular arc graphs [M]. Networks, 1982, 12: 459～467.

5 Tarjan R E. A class of algorithms which require nonlinear time to maintain disjoint sets [J]. Journal of Computer and System Sciences, 1979, 18(2): 110～127.