基于四叉堆优先级队列及逆邻接表的改进型 Dijkstra 算法

陆 锋 卢冬梅 崔伟宏
(中国科学院遥感应用研究所，北京 100101)

摘 要 在深入分析传统 Dijkstra 算法的基础上，提出了利用基于 k 叉堆的优先级队列对算法进行改进的思想，并对 3 种可合并堆进行了比较，从理论上证明了四叉堆在 k 叉堆中的最优性，设计了基于四叉堆优先级队列及逆邻接表、顾及路段方向阻抗的改进型 Dijkstra 最短路径算法，将 Dijkstra 算法复杂度降为 $O(n\log n)$。针对 GIS-T 应用系统的动态特征，提出了 Dijkstra 算法的逆序计算方法，通过构造逆序最短路径树，使算法更具灵活性和实用性。

关键词 最短路径算法 四叉堆 优先级队列 地理信息系统

0 前 言

最短路径问题是交通网络分析中的一个重要问题，也是交通地理信息系统(GIS-T)中的一个研究热点。它是资源分配、路线设计及分析等优化问题的基础。国内外大量专家学者对此问题进行过深入研究。最短路径问题可分为单源最短路径问题及全源最短路径问题两种[1]。其中单源最短路径问题更具有普遍意义，且可为全源最短路径问题提供良好的借鉴方案。单源最短路径问题的算法有很多种，从早期的基于限制条件的深度优先搜索算法[2]、基于有向无环图的动态规划法[3]、基于邻接矩阵的 Dijkstra 算法[1]、到最大相关边法[4]、最大相关点法[5]、基于邻接表的 Dijkstra 算法[6]、A * 算法[7]、基于超图数据结构的深度优先椭圆搜索算法[8,9]等。针对不同的网络特征、应用需求及具体的软硬件环境，各种算法在空间复杂度、时间复杂度、易实现性等方面各具特色。其中，采用贪心及启发策略的 Dijkstra 算法是目前已知的理论上最完善的算法[6]，它以极强的抗差性而得到广泛的普及和应用。

然而，传统的基于邻接矩阵的 Dijkstra 算法，其庞大的网络数据存储需求及算法的空间复杂度自不待言，即使采用邻接表的网络存储结构，在改善存储需求及空间复杂度的同时，对算法的时间复杂度从本质上并没有丝毫改善。而算法的时间复杂度是衡量一个算法效率的最重要因素。虽然基于邻接表的 Dijkstra 算法可以减少部分操作环节，但算法的时间复杂度仍为 $O(n^2)$[1]。也就是说，算法时间复杂度与网络数据的存储结构无关。当然，采用效率较高的邻接表存储结构为算法的进一步优化提供了潜在的可能。

近年来，我们先后承担了城市快速反应系统、智能车辆导航系统及城市安全监控系统等项目的研究及开发工作。其中均涉及到最佳路径的快速算法问题。本文中，在对传统 Dijkstra 算法结构进行分析后，我们提出了利用基于 k 叉堆的优先级队列对算法进行改进的思想，然后对 3 种可合并堆进行了比较，从理论上证明了四叉堆在 k 叉堆中的最优性，设计了基于四叉堆优先级队列及逆邻接表、顾及路段方向阻抗的改进型 Dijkstra 最短路径算法，将传统 Dijkstra 算法复杂度降为 $O(n\log n)$。针对 GIS-T 应用系统的动态特征，我们提出了 Dijkstra 算法的逆序计算方法，通过构造逆序最短路径树，使算法更具灵活性和实用性。本文中提出的算法已在我们承担的车辆智能导航系统及城市安全监控系统中得以实现。

1 传统 Dijkstra 算法描述

Dijkstra 算法由著名数学家 E. W. Dijkstra 于 1959 年首先提出，它是一种采用贪心及启发策略的单源全路径算法，计算结果为一棵以起点为根的最短路径树，如图 1 所示。

收稿日期：1998-11-11；收到修改稿日期：1998-12-23

1.1　基于邻接矩阵的 Dijkstra 算法

基于邻接矩阵的传统 Dijkstra 算法,其核心部分伪码描述如下:[1]

```
D[v0]=0;final[v0]=TRUE;  //  D 为节点的累计阻抗值
①  for (i=1;i<n;i++){    //  主循环,每次求出 v0 到某个节点 v 的最短路径,并将 v 加入集合 S 中
       min=INFINITY;
②      for (w=0;w<n;w++){
           if (! final[w])     // 节点 w 在集合 V-S 中
               if (D[w ]<min){     // 节点 w 距离 v0 更近
                   v=w; min=D[w];
               }
       }
   final[v]=TRUE;  // 将距离 v0 最近的节点 v 加入集合 S 中
③  for (w=0;w<n;w++){  // 更新当前最短路径及距离
           if (! final[w] && (min+c[v][w]<D[w]))     // 修改 D[w],w∈V-S
               D[w]=min+c[v][w];
       }
   }
```

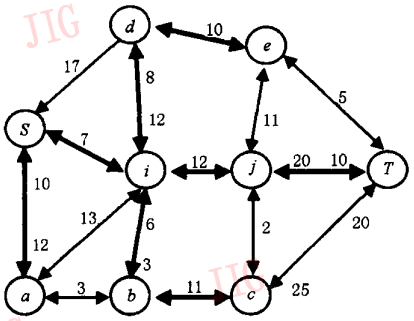

图 1　Dijkstra 算法生成的最短路径图

算法规模可记为 $f(x)=n^2+n^2$,也即时间复杂度为 $O(n^2)$。

值得注意的是,在上述算法描述中,已采用标识数组 $final[i]$取代了传统 Dijkstra 算法中的 OPEN 及 CLOSED 表,以避免集合操作过程。但是,二者在本质上没有任何差异。标识数组的逻辑取值即构成了 OPEN 及 CLOSED 表。这是一个很自然的选择,并非如文献[10]中所说的,对 OPEN 及 CLOSED 表的操作是造成以 Dijkstra 算法为代表的启发式搜索算法庞大存储量及计算量的主要原因,改用节点访问标识域即可节省大量时间。我们可以看出,造成传统 Dijkstra 算法效率不高的主要原因在于算法中的二重循环。当节点总数 n 很大时,此二重循环将耗费大量的计算时间。

1.2　基于邻接表的 Dijkstra 算法

若改用邻接表存储网络数据,则上述算法中①的子循环③可改为:

```
for (w=0;w<v_adj_num;w++) {     // v_adj_num 及 v_adj[w]分别为 v 的邻接节点总数及编号
   if (! final[v_adj[w]] && (min+c[v][v_adj[w]]<D[v_adj[w]]))
       D[v_adj[w]]=min+c[v][v_adj[w]];
}
```

与节点总数 n 相比,某一节点的邻接点数是非常小的。因此,子循环③所耗费的时间可大幅度降低。然而,由于子循环②的存在,使得算法的时间复杂度仍为 $O(n^2)$。

从上述对算法的描述及分析可以看出,欲有效降低 Dijkstra 算法的时间复杂度,关键在于对子循环②进行改造,提高 $V-S$ 集合上的 extract-min 操作的效率。

2　基于四叉堆的优先级队列及改进型 Dijkstra 算法

2.1　优先级队列及 k 叉堆数据结构

上述算法描述中子循环②的唯一目的,就是不断从变化的 $V-S$ 集合中找出下一条最短路径并返

回其父节点指针及累计权值。从中可以看出，由累计权值决定的最短路径选择具有优先程度差异特征。考虑到这一特点，我们可以通过用各节点在算法中不断被松弛的距起点的最短路径值来构造优先级队列，以提高 extract-min 操作的效率。

优先级队列是一种用来维护由一组元素构成的集合 S 的数据结构。作用于优先级队列上的操作主要有 insert、minimum、extract-min、decrease-key 等[6]。实现优先级队列的方法较多，而 k 叉堆无疑是其中十分优秀的一种实现方法。

k 叉堆结构是一种数组对象，可以被视为一棵完全 k 叉树。图 2 即为一个四叉堆优先级队列及其所对应的完全四叉树。堆结构必须满足以下性质：对除了根节点以外的每个节点 i，有

$A[parent(i)]\leq A[i]$ 或 $A[parent(i)]\geq A[i]$

即某个节点的值不小于(或不大于)其父节点的值。这样，堆中的最小(最大)元素就存放在根节点中。且每一节点的子树中的节点值都不小于(或不大于)该节点的值。

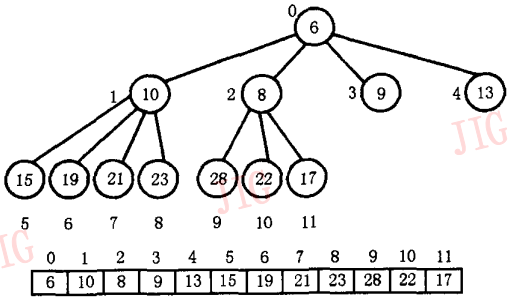

图 2 四叉堆及所对应的四叉树

因为具有 n 个元素的 k 叉堆是基于一颗完全 k 叉树的，则其高度为 $\lfloor \log n \rfloor$† 。对于 k 叉堆的操作，其作用时间至多与树的高度成正比，为 $O(\log n)$‡ 。因此，若采用基于 k 叉堆实现的优先级队列来存储 Dijkstra 算法中子循环③所扩展出的 D 值及完成对 D 值的松弛操作，则可大幅度降低算法的时间复杂度，提高算法效率。

2.2 k 叉堆操作时间分析及 k 值选定

虽然对于一定大小的整数 k，k 叉堆操作的运行时间界均为 $O(\log n)$，但是，由于 k 值的不同，使得堆操作中的赋值、比较及交换 3 种基本操作的次数有很大差异。因此很有必要对 k 叉堆的操作时间进行深入分析，寻求最合适的 k 值，使所涉及到的堆操作的运行时间最少。

在 Dijkstra 算法所涉及到的复杂度为 $O(\log n)$ 的 3 种 k 叉堆操作中，其主体均为 heapify 操作。heapify 操作用来保持 k 叉堆的性质，其伪码描述如下：

```
heapify (A,i)
{
    c1=child1(i); c2=child2(i); …; ck=childk(i)
                                  // 对子节点下标赋值;
    // 在节点 i 与 k 个子节点中寻找最小值
    if (c1≤heap-size[A] && A[c1]A[i])
                        smallest=c1;
    else smallest=i;
    if (c2≤heap-size[A] && A[c2]A[smallest])
                        smallest=c2;
    …
    if (ck≤heap-size[A] && A[ck]A[smallest])
                        smallest=ck;
    // 如果需要,则交换元素以维护堆性质
    if (! smallest==i){
        swap(A[i], A[smallest]);
        heapify(A, smallest);
    }
    else return;
}
```

由以上伪码可以看出，在最坏情况下，heapify 操作需进行 $(k+1)\log_k n$ 次比较，$2k\log_k n$ 次赋值，$\log_k n$ 次交换。其中一次交换操作可等价于三次赋值操作。若将一次赋值操作时间记为 t_1，一次比较操作时间记为 t_2，则在最坏情况下，heapify 操作所需时间共为：

$$\begin{aligned} S_t &= t_1(2k+3)\log_k n + t_2(k+1)\log_k n \\ &= \log_k n[(2k+3)t_1 + (k+1)t_2] \\ &= \frac{\ln n[(2k+3)t_1 + (k+1)t_2]}{\ln k} \end{aligned} \tag{1}$$

欲使 S_t 取得最小值，(1)式对 k 求导，得

$$S_t = \frac{\ln n[k(2t_1+t_2)\ln k - (2k+3)t_1 + (k+1)t_2]}{k\ln^2 k} = 0 \tag{2}$$

(2)式化简后得

$$\ln k = 1 + \frac{3t_1 + t_2}{k(2t_1 + t_2)} \tag{3}$$

† 因 $\log_k n = \log_2 n / \log_2 k$，为描述方便，在不影响分析的前提下，本文中一律以 $\log n$ 取代 $\log_k n$。

‡ 为方便比较，在不至引起歧义的前提下，本文中统一用渐近上界符号 O 取代渐近上确界符号表示算法的时间复杂度。

不失一般性,设 $t_1=ct_2(c>0\ \&\ c\in R)$,则(3)式可化为:

$$\ln k=1+\frac{3c+1}{k(2c+1)} \tag{4}$$

(4)式的最佳整数解为 4,也即当 $k=4$ 时,S_t 取得渐近最小值。

从形式上分析,四叉堆与二叉堆在最坏情况下所需的赋值操作次数一样,但二叉堆所需的交换操作次数是四叉堆的两倍,比较次数也比四叉堆略多。按照 Ghamdi 的研究结果,交换操作的数量是影响排序类算法效率的主要因素[12]。由此也可看出四叉堆较之二叉堆所具有的优越性。

在 heapify 操作中对节点下标的计算,四叉堆与二叉堆一样,通过二进制的位操作,最多用两条二进制指令即可得到某节点所对应父节点及各子节点下标值,而 k 值的其它实现,大多至少需要 3 条指令才能完成。

2.3　可合并堆 Dijkstra 算法适应性比较分析

与 k 叉堆同属可合并堆,并可用于优先级队列的还有二项堆及 Fabonacci 堆等。但是,二项堆与 Fabonacci 堆的实现远较 k 叉堆复杂。此外,由于 Dijkstra 算法中不涉及 heap-union 操作,对于所涉及到的其它各项操作,二项堆与 k 叉堆均有着同样的最坏情况运行时间界[6];而 Fabonacci 堆虽然对于 heap-insert 操作及 heap-decrease-key 操作有着更好的时间界,但是按照 Goldberg 等人的研究,Fabonacci 堆在减小 heap-insert 及 heap-decrease-key 操作时间界的同时,却增大了 heap-extract-min 操作的复杂度,并且增大了它的常数因子[13]。这使得 heap-insert 及 heap-decrease-key 操作时间的减小并不足以令 Fabonacci 堆比 k 叉堆表现得更好。在 Dijkstra 算法中,针对交通网等稀疏图,heap-decrease-key 的最大操作次数 m(弧段数)并非远大于 n(节点数),使得 Fabonacci 堆并不能发挥其优越性。此外,Goldberg 等的研究还表明,在假定节点入度弧段长度服从同一分布并相互独立时,k 叉堆的 heap-decrease-key 操作的平均时间界仅为 $O(n\log n\log\lceil(1+m)/n\rceil)$,远小于通常所认为的 $O(m\log n)$。所假设前提是一个高概率事件,当 $n\to\infty$ 时,其概率趋近于 1。综上所述,我们可以看出,k 叉堆是三者之中最适合交通网络 Dijkstra 算法的可合并堆。

2.4　基于邻接表及四叉堆优先级队列的 Dijkstra 算法

利用四叉堆优先级队列,可对基于邻接表的 Dijkstra 算法进行改造。改造后算法核心部分伪码描述如下:

```
D[v0]=0; final[v0]=TRUE;
make-heap(A);
for (i=0; i<v0_adj_num; i++)   // 将 v0 邻接节点对应 D 值插入堆中
heap-insert(A,D[v0_adj[w]]);
for (i=1; i<n; i++){
    heap-extract-min(A,v);   // 从堆中抽出最小 D 值,标识为已处理节点
     final[v]=TRUE;
     for (w=0;w<v_adj_num;w++){
            if (! final[v_adj[w]] && (D[v]+c[v][v_adj[w][<D[v_adj[w]])){
            if (D[v_adj[w ]]==INFINITY){   // 未插入堆中的节点
                        D[v_adj[w]]=D[v]+c[v][v_adj[w]];
                        heap-insert(A,D[v_adj[w]]);
                  }
                  else {   //   已插入堆中但需松弛节点
                        D[v_adj[w]]=D[v]+c[v][v_adj[w];
                        heap-decrease-key(A,pos,D[v_adj[w]]);
                  }
            }
     }
}
```

2.5 算法时间复杂度分析

由上述算法的伪码描述可知,原先的子循环①已被优先级队列的 heap-extract-min 操作所取代,其最坏情况下的运行时间为 $O(1)+O(\log(n-1))=O(\log n)$。子循环③中,从形式上分析,虽然一个节点可能经多次松弛,但任一条边在整个算法中仅仅处理一次,即此子循环运行时间最多为 $O(m\log n)$。也就是说,在整个算法运行过程中,在此循环花费时间(包括主循环)共计 $m\log n$。(按照 Goldberg 等人的研究,在高概率前提下,heap-decrease-key 操作的平均时间界仅为 $O(n\log n\log[(1+m)/n])$[12],此处为时间复杂度的形式化分析。)对于交通网络之类的稀疏图,有 $1<m/n<2$,故整个算法运行时间为 $O(n\log n)+O(m\log n)=O(n\log n)$。而传统的基于邻接矩阵或邻接表的 Dijkstra 算法,其时间复杂度为 $O(n^2)$。可见,基于邻接表及优先级队列的改进型 Dijkstra 算法,其运行效率已大为改善。

3 交通网络数据的逆邻接表表达及最短路径树的逆序计算

Dijkstra 算法中的全路径计算是保证单对节点间路径最佳的关键因素。它保证了 Dijkstra 算法所具备的极强的抗差性[11]。只要网络中不存在负权边,Dijkstra 的全路径算法总能找到单对节点间的最佳路径。即使在最坏的情况下,从渐近意义上看,目前还没有比最好的单源算法更快的算法来解决单对节点间最短路径问题[6]。

在诸如车辆调度、导航、路线规划等 GIS-T 应用中,由于交通网络的动态变化特征,使得由一般计算顺序所产生的最短路径树并不能完全满足需要。如在车辆导航系统运行中,当目标运行至某一节点 i 时,发现路段 l_{ij} 发生交通堵塞,目标不得不选择其它线路运行。这时,已计算出的最短路径树即宣告作废,只能在任意选择的 i 的邻接节点处重新计算。这势必使应用系统运行效率大为降低。

在本文中,针对交通网络的动态变化特征及方位特征,我们提出 Dijkstra 算法的逆序运行方法,将计算顺序由 $S\rightarrow i,i+1,i+2,\cdots T\cdots i+n$ 改为 $T\leftarrow i,i+n,i+n-1,\cdots S\cdots i$,即构造以终点 T 为根的逆序最短路径树,使起点 S 位于最短路径树的枝节点上。它绝不同于以 T 为起点运行 Dijkstra 算法所得到的最短路径树。考虑交通网络的方向性,以 T 为起点所得到的最短路径树对 S、T 间的最短路径问题是无用的。而我们提出的逆序 Dijkstra 算法所得到的以终点 T 为根的逆序最短路径树如图 3 所示。

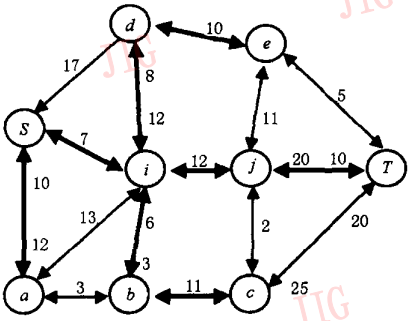

图 3 以 T 为终点的逆序最短路径树

按照我们提出的逆序 Dijkstra 算法,得到逆序最短路径树后,网络中每一节点到终点 T 的最短路径均已知。这样,当路段 l_{ij} 发生交通堵塞时,只需在 i 节点的出邻接点集合 P 中选择满足 $\min(D[k]+c_{ik}\mid k\in P,\&\ k\neq j)$ 的节点作为前至节点即可,不需要以各邻接点为根重新计算最短路径树,大大提高了系统运行效率。

采用逆序 Dijkstra 算法,需要以逆邻接表方法存储交通网络的拓扑信息,即对每个顶点 v_i 建立一个链接以 v_i 为头的弧的表。图 3 中的 S 与 T 点的逆邻接表如图 4 所示。

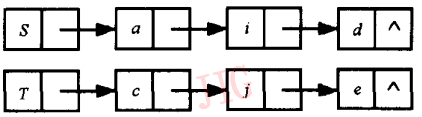

图 4 S 与 T 点的逆邻接表

基于逆邻接表及四叉堆优先级队列的逆序 Dijkstra 算法,其核心部分伪码似于 2.4 中所给出的伪码,这时 v_adj_num 为节点 v 的入度。另外,需将 $c[v][v\_adj[w]]$ 置换为 $c[v\_adj[w]][v]$,它表示路段 $l_{v\_adj[w]v}$ 的权重。

4 实 例

本文中提出的改进型 Dijkstra 算法已在城市安全监控系统及车辆智能引导系统中得以实现。系统交通网络空间数据及拓扑数据结构如图 5 所示。

上述数据结构充分表达了交通网络的空间、属性及拓扑信息,并为顾及路段禁行、单行线及同一路

段不同方向阻抗的改进型 Dijkstra 最短路径算法提供了良好的基础。其中道路方向阻抗可表达为路段长度与类型的函数,同一条路段不同方向可具有不同阻抗值。

本文所提出的算法已在 MS VC++编译环境下完成。以北京市为例,由街区道路构成的交通网络共包括 12800 个节点及 17800 条路段,在这样一个庞大的交通网络上,以道路方向阻抗为权值,任意选择起始及终止节点,运行本文提出的改进型 Dijkstra 最短路径算法,在 Pentium Pro 200 微机上不超过 1 秒钟,比同一数据在 SUN Ultra1 ARC/INFO V7.12 上运行速度提高 7—8 倍。与文献[10]4000 个节点的单源最佳路径计算需耗时 2 分钟相比,无疑是一个质的飞跃,完全可以满足大规模交通网络应用需求。

实践表明,本文提出的算法能够大幅度降低 Dijkstra 算法的时间复杂性,提高系统运行效率,在 GIS-T 应用系统的开发中具有广阔的应用前景。

(a) 交通网络空间数据结构

Arc-ID	Vertex Coordinate List
1	x_1y_1, x_2y_2,…
2	x_1y_1, x_2y_2,…
…	…
m	x_1y_1, x_2y_2,…

(b) 节点位置及拓扑数据结构

Node-ID	X Coord.	Y Coord.	Adj. Number	Incoming Node List	Incoming Arc List
1	x_1	y_1	adj_num$_1$	in_n_1,in_n_2,…	in_a_1,in_a_2,…
2	x_2	y_2	adj_num$_2$	in_n_1,in_n_2,…	in_a_1,in_a_2,…
…	…	…	…	…	…
n	x_n	y_n	adj_num$_n$	in_n_1,in_n_2,…	in_a_1,in_a_2,…

(c) 路段属性数据结构

Arc-ID	Name	Head Node.	Length	Class	Head-End Imped.	End-Head Imped.
1	name$_1$	hn$_1$	l$_1$	c$_1$	he_imp$_1$	eh_imp$_1$
2	name$_2$	hn$_2$	l$_2$	c$_2$	he_imp$_2$	eh_imp$_2$
…	…	…	…	…	…	…
m	name$_m$	hn$_m$	l$_m$	c$_m$	he_imp$_m$	eh_imp$_m$

图 5　GIS-T 交通网络空间数据及拓扑数据结构

参 考 文 献

1 严蔚敏,吴伟民编著. 数据结构(C 语言版). 北京:清华工学出版社,1997.

2 Stig Nordbeck, Bengt Rystedt. Computer Cartography Shortest Route Programs. Sweden: The Royal University of Lund, 1969.

3 张益新,沈雁编著. 算法导论. 长沙:国防科技大学出版社,1995.

4 徐立华. 求解最短路问题的一个计算机算法. 系统工程,1989,7(5):46～51.

5 龚洁辉. 最短路径算法的改进及面向对象的实现方法. 解放军测绘学院学报,1998,15(2):121～124.

6 潘金贵,顾铁成,曾　俭等编译. 现代计算机常用数据结构和算法. 南京:南京大学出版社,1994.

7 Rune. The A* algorithm: Comparison to other common path-finders. http://www.cs.auc.dk/～rune/ FE1101/litt/Astar/PathFinders.html, 1997.

8 陈行星,崔伟宏. 城市快速反应系统实验研究. 环境遥感,1996,11(3):227～233.

9 崔伟宏. 空间数据结构研究. 北京:中国科学技术出版社,1995.

10 徐业昌,李树祥,朱建民等. 基于地理信息系统的最佳路径搜索算法. 中国图象图形学报,1998,3(1):39～42.

11 Wicks J. Shortestpaths. http://www.northpark.edu/acad/math/courses/Math-1030/WebBook/GraphTheory/Dijkstra.html

12 Al-Ghamdi K S. Analysis of quick sort and heap sort. http://cs.fit.edu/～wds/syl/cse5081/cse5081.html

13 Goldberg A V, Targan R E. Expected performance of Dijkstra's shortest path algorithm. http://cs-tr.cs.cornell.edu:80/Dienst/UI/2.0/Search, 1996.

陆　锋　1970 年生,1991 年毕业于武汉测绘科技大学航测与遥感系摄影测量与遥感专业,现于中国科学院遥感应用研究所攻读博士学位。主要从事地理信息系统及全球定位系统的应用研究。

卢冬梅 1969 年生,1992 年毕业于北京计算机学院软件工程专业,现为中国科学院遥感应用研究所助理研究员。研究方向为全球定位系统应用系统开发。

崔伟宏 1962 年毕业于莫斯科测绘学院,现为中国科学院遥感应用研究所研究员,博士生导师。主要研究方向为地理信息系统,空间数据结构,空间决策支持系统,全球定位系统应用及农业信息网络等。

Improved Dijkstra Algorithm Based on Quad-Heap Priority Queue and Inverse Adjacent List

Lu Feng, Lu Dongmei and Cui Weihong

(*The Institute of Remote Sensing Applications, Chinese Academy of Sciences, Beijing* 100101)

Abstract An improved Dijkstra algorithm is set forward on the basis of priority queue based on quad-heap. It is proved in this paper that quad-heap is the optimum among k-ary heap. The presented algorithm decreases the complexity of conventional Dijkstra algorithm to $O(\log n)$. Considering the dynamic characters of transportation applications, an inverse running sequence is presented based on the inverse adjacent list data structure. It is more suitable for the dynamic applications of GIS-T.

Keywords Shortest path algorithm, Quad-heap, Priority queue, Geographic Information Systems

(上接第 1033 页)

MapGIS 大多数模块比较成熟,实用性强,在大数据量管理和空间分析、网络分析和综合应用等方面表现优秀。缓冲区分析、全库查询性能优于国外同类软件,已经占有相当的国内市场,在基础软件测评中名列第一。特向市场推荐;

GeoStar 软件功能齐全,可以在 GIS 应用工程中使用,在数字高程模型处理方面表现优秀,可用性优于国外软件。在基础软件测评中名列第二。特向市场推荐;

MapEngine 实现了全组件化,有利于二次开发,有自己的特色,综合应用表现良好,特提出表扬。

AM/FM 应用开发平台 **Grow** 实现了多级服务器、多用户协同工作,已经用于电力、消防、燃气、交通等领域,在本次测评中表现突出,特向市场推荐;

下列软件实际应用效果良好,在市场上有较大的影响,在本次测评中表现突出,特向市场推荐:

图形化电网综合信息管理系统 **EFGIS**;

MapGIS 供水管网管理系统;

MapGIS 通信管网管理系统;

地图出版系统"方正智绘"、**MAPCAD** 彩色地图编辑出版系统;

扫描数字化软件 **GeoScan**、**EPSCAN2000**;

数字高程模型处理、分析和显示软件 **GeoTin** & **GeoGrid**;

电子平板软件 **EPSW2000**;

Internet 上信息发布、浏览和查询软件 **GeoSurf**、**AFInternetGIS**;

数字正射影像处理软件 **PhotoMapper**;

电子出版物"**北京通 V3.0**"。

测评结果再次说明了市场和应用是技术发展的原动力。本次参加测评的软件绝大多数都是在实际使用中经受考验的软件产品,适合中国的行业规范和用户使用习惯,出现了一批上乘之作。

测评结果同时也说明了科技部坚持发展 GIS 要走产业化道路的决策是正确的,科技功关管理坚持"引入竞争机制,坚持滚动发展"的方针是正确的。软件水平在竞争中得到了很快的提高。出现了不少值得称道的软件产品。

每年一度的 GIS 软件测评起到了良好的导向作用。在技术方面,"组件化"、"全关系"、"三层结构"等技术已经在参测软件中使用,代表发展方向的统一存放空间数据的国产数据库软件今年也参加了测评;在市场方面,企业注重提高商品化程度和技术服务,在与国外软件的竞争中快速增强实力,市场份额迅速扩大。

我国的 GIS 软件产业已经起步。我们相信,只要坚持走"以用立业"的道路,走技术创新和跨越道路,坚持产业化方向,国产 GIS 软件的水平一定可以在技术创新过程中不断提高,我国的 GIS 软件产业一定可以在市场竞争中发展壮大。