文章编号:1001 - 2486(2004)04 - 0051 - 06

保持连通的边缘细化算法*

谭郁松,周兴铭
(国防科技大学计算机学院,湖南 长沙 410073)

摘 要:传统的图像边缘检测算子一般都只能得到多像素宽边缘,这为后续的图像处理带来了一定困难。结合边缘走向趋势的估计技术以及对连通关键点判断的方法,提出了一种新型的边缘细化算法——保持连通的边缘细化算法。该算法能在保持边缘原有信息(连通和走向)的前提下,以较小的计算开销,给出理想的或是可接受的单像素宽细化结果。

关键词:图像边缘细化;最长路径;关键点
中图分类号:TP391 文献标识码:A

Connectivity Preserved Edge Thinning Algorithm

TAN Yu-song, ZHOU Xing-ming
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: The traditional image edge detectors can only get multiple-pixel-wide edges usually, thus it is difficult to operate by post-processing methods. The paper presents a novel edge thinning algorithm named as connectivity preserved edge thinning algorithm, which combines the detecting strategy of the connectivity critical point with the estimating method of the edge direction. It can preserve the origin edge information such as connection and direction. Moreover, it gets ideal or acceptable single-pixel-wide edges with lower computing cost.

Key words: image edge thinning; longest path; critical point

图像的边缘信息是其主要内容,边缘描述了图像中所包含物体的轮廓,表达了物体之间的关系。边缘处理是图像处理领域中的基础技术,它为更复杂的图像分割、区域增长、目标识别、模式识别等操作奠定了重要基础。但是在传统的边缘检测算法中,所得到的边缘可能很‘厚”,远非理想中的单像素宽,这为后续的图像处理引入了较大的困难,如边缘曲线的拟合等。

本文提出了保留连通的边缘细化算法(Connectivity Preserved Edge Thinning Algorithm,CPETA),该算法在局部区域上寻找最长路径用以粗略估计边缘的走向趋势,利用关键点的思想来保证边缘的连通。试验结果令人满意。

1 相关研究

正由于边缘细化算法较为重要,因此其一直以来都得到学者的关注,并已取得了一定的研究成果。现有的边缘细化算法可粗略地分为两大类:基于二值边缘图像的方法以及基于数学形态学的方法。前者见 Zhang[1]、Hall[2]、Holt[3]、Park[4]、Prewer[5] 等的工作,而基于数学形态学的方法见 Lee[6]、Lobregt[7]、Bernard[8] 以及车武军[9]等的工作。

二值边缘图像便是只包含是否归属于边缘点的信息,真为 1,假为 0。在此类方法中,细化效果较好且速度较快的算法是文献[2,3]中所给出的 HSCP 算法。因此,该算法也选用为本文的实验参考算法。首先给出图像中像素点的 8 连通域示意图,如图 1 所示。

* 收稿日期:2004 - 03 - 08
作者简介:谭郁松(1976—),男,博士生。

则 HSCP 算法可以简述如下：

步骤 1：针对所有的边缘点 P_0，若满足如下条件，则判断其为可删除的。

(1) 其 8 连通域中的边缘点数 $E(P_0)$ 为 $2 \leqslant E(P_0) \leqslant 6$；

(2) P_0 的 8 连通域中包含且只包含一个 4 连通边缘点。

步骤 2：遍历所有的可删除点，若满足如下条件之一，则保留；否则删除。

(1) P_2、P_6 为边缘点，而 P_4 为可删除的；

(2) P_4、P_8 为边缘点，而 P_6 为可删除的；

(3) P_4、P_5、P_6 均为可删除。

P_1	P_2	P_3
P_8	P_0	P_4
P_7	P_6	P_5

图 1 8 连通域示意图

Fig.1 Illustration of 8-connected field

从中不难发现，此类的典型算法是在每次迭代过程中均需要两次遍历，首先需要遍历全体边缘点，依某种判断条件，判断出边缘点的可删除性；而后再次遍历这些全体可删除边缘点，再次判断这些点是不是确实可删除。尽管每次的迭代对象不尽相同，算法的计算开销依然较大。

Park[4] 的方法是在二值边缘图像的基础上，结合边缘梯度的方向角来判断边缘的发展趋势。该算法能给出较好的结果，但需要引入大量的浮点运算，开销太大。而基于数学形态学的边缘细化，则主要是依据骨架(Skeleton)的思想。此类方法能给出边缘的大概形状，连通性保持效果好，但细化结果也并不太理想。

2 边缘细化算法

本文认为，一个性能良好的边缘细化算法应该具有如下三个性质：

(1) 单像素宽细化结果

理想的算法应该能给出单像素宽的细化结果。

(2) 保持边缘的原有信息

边缘细化结果不能改变边缘的原有信息，即边缘的连通和走向趋势，因为这些信息是进行图像理解、目标识别的重要依据。

(3) 计算开销小

能被实际广泛运用的算法自然是计算开销小、执行效率高的方法。

本小节则将介绍一种有效的边缘细化算法——保留连通的边缘细化算法。该算法的主要思想是结合边缘走向趋势的估计技术以及对连通关键点判断的方法，在保持边缘原有信息的前提下，给出理想的或是可接受的单像素宽细化结果。并且该算法在每次迭代中，只需扫描边缘点一次，从而使得计算开销较小。本文假设所处理的对象均为二值边缘图像。

2.1 边缘信息的保持

若要保持边缘连通，就需判断出边缘的原有连通情形以及连通关键点。所谓关键点(Critical Point)就是 8 连通域中的‘连通枢纽’边缘点，关键点的删除将会破坏 8 连通域中其他边缘点之间的连通。此处的连通表示成 0°、90°、45°、135°方向上相邻点的关系。

图 2 给出了如何进行连通关键点判断的基本方法。

不难发现，算法 1 主要是遍历 8 连通域，试图寻找连接所有非 P_0 的边缘点之间的路径。若能找到，则表示 P_0 不会影响其他边缘点之间的连通；否则，P_0 将定义为关键点。

现在需要考虑如何对边缘的走向趋势进行有效的估计。显然，最正确的方法便是对每个像素点都进行梯度的计算，但是这需要引入大量的浮点计算，开销很大。因此，本文将采用一种基于寻找 8 连通域中最长路径的简单估计方法。最长路径在一定程度上表示了整个边缘的行走方向，尽管它只是局部信息，但也能用于边缘走向的大致估计。并且该方法不涉及到大量、复杂的浮点数值运算，开销较小。

步骤 0:设定 $i=1$,$nPathSum=0$;
步骤 1:循环进行步骤 2 至步骤 4,直至 $i>8$;
步骤 2:判断 P_i 是否为边缘点;
步骤 3:若 P_i 是为边缘点,则判断 P_{i+1}是否也为边缘点:
步骤 3.1:若 P_{i+1}是边缘点,则 $nPathSum++$,且 $i++$;
步骤 3.2:若 P_{i+1}不是边缘点,且 $i\%2=0$,则判断 P_{i+2}是否也为边缘点:
步骤 3.2.1:若 P_{i+2}是边缘点,则 $nPathSum++$,且 $i+=2$;
步骤 3.2.2:若 P_{i+2}不是边缘点,则 $i++$;
步骤 3.3:若 P_{i+1}不是边缘点,且 $i\%2==1$,则 $i++$;
步骤 4:若 P_i 不是边缘点,则 $i++$;
步骤 5:判断以 P_0 为中心的 8 连通域中的边缘点数目 $nEdgePointNum$,其中需要包括在本次迭代中已被删除的边缘点;
步骤 6:若($nEdgePointNum-1==nPathSum$)$\cap$($nEdgePointNum>1$)$==True$,则判断 P_0 是本 8 连通域中的关键点,返回 $True$;否则返回 $False$。

图 2 算法 1:关键点判断

Fig.2 Detection of the critical point

如图 3 所示算法便能给出针对 8 连通域进行最长路径的判断方法。

步骤 1:寻找包含 $P0$ 的最长路径,即沿着 $P_1-P_0-P_5$、$P_2-P_0-P_6$、$P_3-P_0-P_7$、$P_4-P_0-P_8$ 等四条路径分别求路径的长度。并假定最长路径的长度为 L。
步骤 2:沿着 $P_1-P_2-P_3$、$P_3-P_4-P_5$、$P_5-P_6-P_7$、$P_7-P_8-P_1$ 等四条路径分别进行如下操作:
步骤 2.1:求出不包含在本次迭代中已被删除的边缘点的路径长度,假定路径的长度为 L';
步骤 2.2:如果 $L'\geqslant L$,则判断 P_0 不在 8 连通域的最长路径中,立即返回 $False$。
步骤 3:如果 $L=2$,则沿着 P_2-P_4、P_4-P_6、P_6-P_8、P_8-P_2 等四条路径分别进行如下操作:
步骤 3.1:求出不包含在本次迭代中已删除的边缘点之后的路径长度,假定路径的长度为 L';
步骤 3.2:如果 $L'\geqslant L$,则判断 P_0 不在 8 连通域的最长路径中,立即返回 $False$。
步骤 4:返回 $True$。

图 3 算法 2:最长路径的判断

Fig.3 Detection of the longest path

注意到,算法 2 在判断包含 P_0 的最长路径时,需要综合考虑在本次迭代中已被删除的边缘点;而判断不包含 P_0 的最长路径时,却不考虑在本次迭代中已被删除的边缘点。该策略有助于保持边缘点,但缺点是有可能增加边缘点细化的迭代次数。

2.2 CPETA 算法描述

在算法 1 和算法 2 的基础上,不难给出如图 4 所示的 CPETA 算法流程。

算法 3 的步骤 3.1 和步骤 3.2 需遍历同一个 8 连通域,因此可以在一次遍历中并发实现两种功能,也应该并发实现,从而避免对同一个 8 连通域的多遍扫描,降低计算复杂度。

注意到,CPETA 算法的步骤 3 中需要判断 $2\leqslant E(P_0)\leqslant 6$。当 $E(P_0)=1$ 时,自然需要保留 P_0 点;而在 $E(P_0)\geqslant 7$ 时,算法认为 P_0 点处于边缘簇集区域,仅从 3×3 的小窗口的信息中不足以判断边缘的走向,因此一律保留,以期在后续迭代中能进行正确判断。

3 试验结果

本文给出了 CPETA 算法以及 HSCP 算法之间细化性能比较,所给出的试验图片具有一定的典型性,其他的文献[1~6]也基本上采用了类似形状的测试集。

图 5(a)、(b)、(c)给出了横直线、纵直线、45°斜边、135°斜边以及 160°斜边的细化结果比较。可以发

步骤 0 遍历所有边缘点，假设现处理边缘点 P_0。
步骤 1 获取边缘点 P_0 的 8 连通域中的边缘点数目 $E(P_0)$，其中统计数目不能包括中心点，但需要包含在本次迭代中已被删除的边缘点。
步骤 2 如果 $E(P_0)=0$，则认为该点是孤立点，删除之。
步骤 3 如果 $2 \leqslant E(P_0) \leqslant 6$，则执行如下操作：
步骤 3.1 判断 P_0 是否处于 8 连通域中最长路径上，即判断存在性。如为真，则 *bLongestPath = True*；否则 *bLongestPath = False*；
步骤 3.2 判断 P_0 是否是本 8 连通域中的关键点，若为关键点，则 *bCriticalPont = True*，否则 *bCriticalPont = False*；
步骤 3.3 若 *bLongestPath = True* 且 *bCriticalPont = True*，则删除该边缘点。

图 4 算法 3：CPETA 算法

Fig.4 CPETA Algorithm

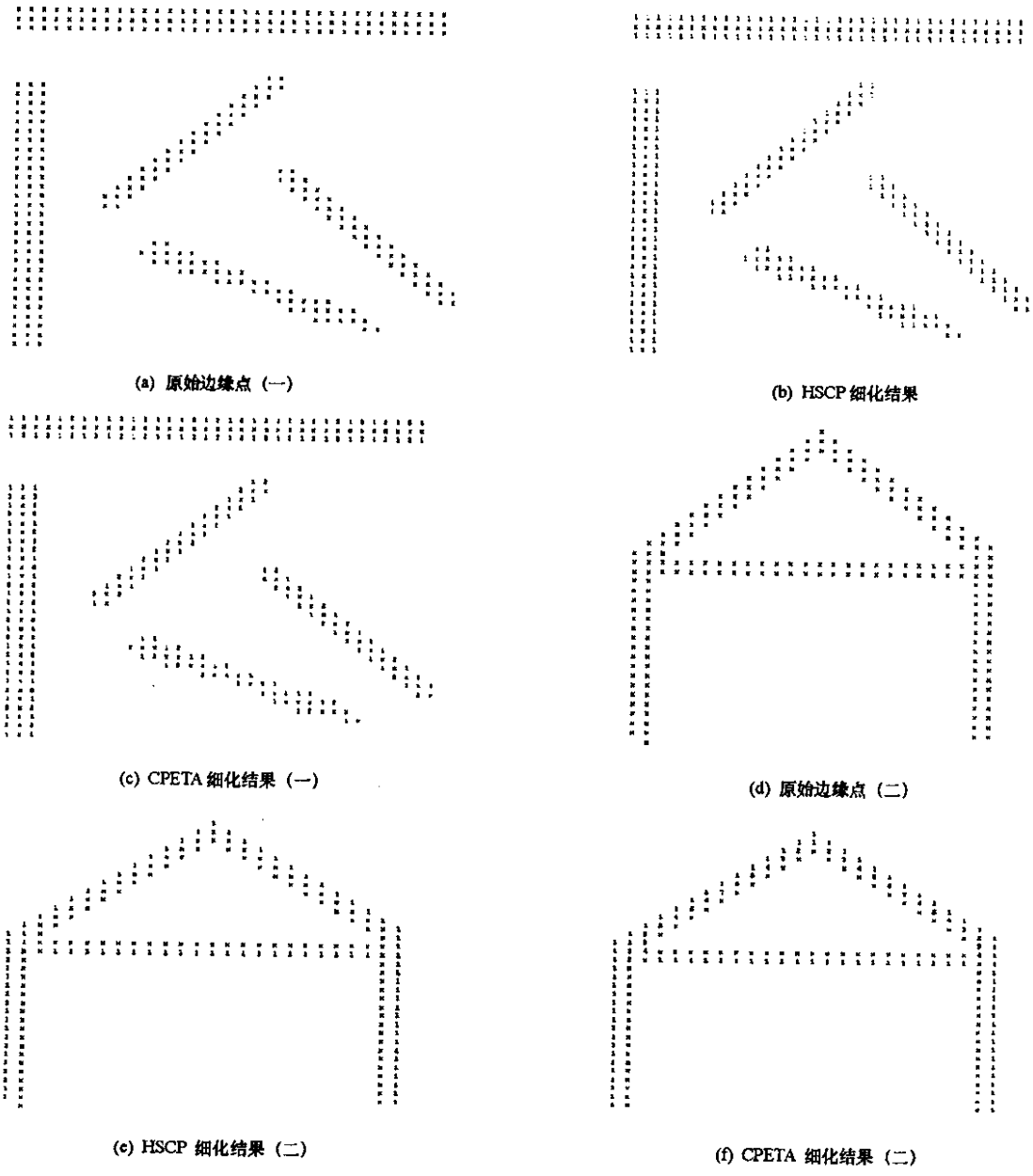

图 5 两种情形的细化结果比较

Fig.5 Two edges' thinning results

现,对于横直线、纵直线,HSCP 算法可以给出理想的细化结果;但是对于 45°斜边和 160°斜边,细化结果不理想;而对于 135°斜边,则给出了不可接受的结果。但是 CPETA 算法对于全部情形都能给出理想的或是满意的单像素宽细化结果。图 5(d)、(e)、(f)给出了"A"型边缘点的细化结果对比图,该图主要综合比较纵、横直线,以及多像素宽的斜线的处理结果。可以发现,HSCP 算法对于多像素宽的斜线细化效果较差,而 CPETA 算法则能得到正确结果。

图 6(a)、(b)、(c)给出了"K"型边缘点的细化结果对比图。该图主要综合比较对双像素宽的纵直线,以及斜线的处理结果。从 HSCP 的细化结果图中可以发现,HSCP 对于斜线的处理结果不令人满意。例如两条斜边细化后长度大大缩减,而 CPETA 算法则能给出可接受的细化结果。图 6(d)、(e)、(f)则是对于一个较为综合的处理结果的比较。可以发现,较 HSCP 算法而言,CPETA 算法的细化结果更令人满意。

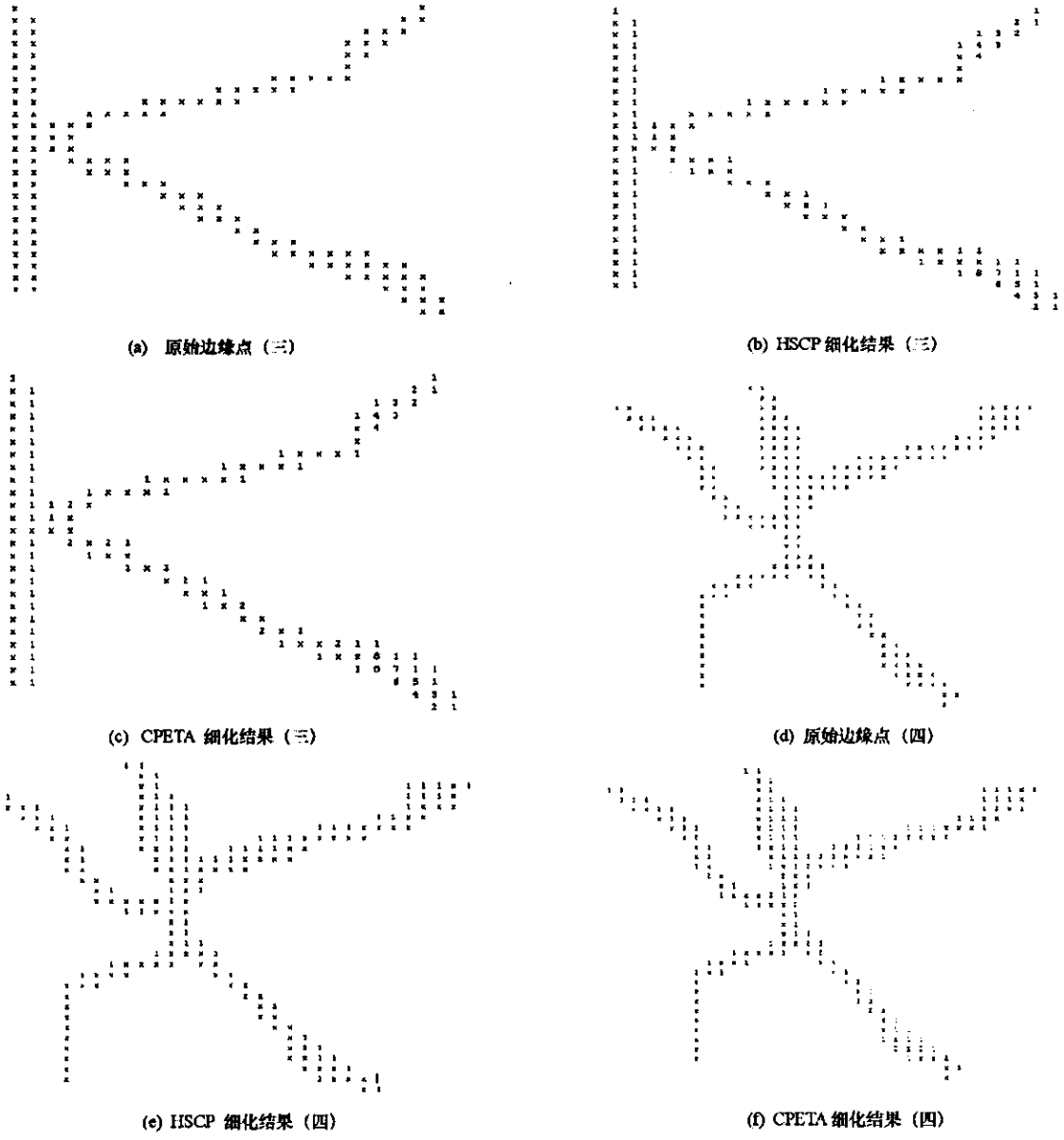

图 6 另外两种情形的细化结果比较

Fig.6 More edges' thinning results

因此,CEPTA 算法对于大部分的边缘情形均能给出较 HSCP 算法更好的边缘细化结果。

但是 HSCP 算法能在多数情形下仅用一次迭代便得到结果，而 CPETA 算法则只能在第一次迭代中处理绝大部分的边缘点，还需要对少量的边缘点进行 3 次迭代。但是需要注意的是，HSCP 算法在每次迭代中需要遍历边缘点两次，而 CPETA 算法仅仅需要遍历一次。因此 CPETA 整体的运算开销并不一定比 HSCP 算法要差。表 1 就给出了两种算法之间的计算开销比较，试验用机的 CPU 为 Intel Cerlon 1.7GHz，内存为 256 DDR266。图像大小为 512 × 512。

表 1 HSCP 算法和 CPETA 算法的计算速度比较

Tab.1 Comparison of HSCP and CPETA 's computation cost

	HSCP 算法(ms)	CPETA 算法(ms)
综合边缘	10	12
“K”型边缘	7	5
“A”型边缘	7	8
人型边缘	15	17

由上表可知，两种算法之间的运算性能相当。特别是当图像边缘的斜边较多时，两者算法性能相当。但图像边缘比较规整，特别是主要包含纵、横边缘点较多时，CPETA 算法所需迭代次数较多，因此运算开销较大。

4 结 论

本文主要提出了一种新型的保留连通的边缘细化的算法—CPETA 算法。该算法能在保持边缘原有信息(连通和走向)的基础上，对大部分的边缘类型给出理想或是令人满意的单像素细化结果。该算法在每次迭代中只需要遍历边缘点一次，故计算开销也较小。对比于传统算法，本算法的实验结果令人满意。

参 考 文 献：

[1] Zhang T Y, Suen C Y. A Fast Parallel Algorithm for Thinning Digital Patterns [J]. Communication of the ACM, March 1984, 27(3): 236 – 239.

[2] Hall R W. Fast Parallel Thinning Algorithms: Parallel Speed and Connective Preservation [J]. Communication of the ACM, Jan. 1989, 32(1): 124 – 131.

[3] Holt C M, et al. An Improved Parallel Thinning Algorithm [J]. Communication of the ACM, Feb. 1987, 30(2): 156 – 160.

[4] Park Jung-Me, et al. A New Gray Level Edge Thinning Method[R]. Univ. of Alabama, 2000.

[5] Prewer D, Kiteben L. A Fast Table Method for Edge Thinning and Linking[R]. Tech. Report, WP1999/9, The University of Melbourne, 1999.

[6] Lee T C, Kashyap R L. Building Skeleton Models via 3-D Medial Surface/Axis Thinning Algorithms [C]. CVGIP: Graphical Models and Image Processing, 1994, 56(6): 462 – 478.

[7] Lobregt S, Verbeek P W, Groen F C A. Three-dimensional Skeletonization: Principle and Algorithm [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1980, 2(1): 75 – 77.

[8] Bernard T M, Manzanera A. Improved Low Complexity Fully Parallel Thinning Algorithm [C]. Proc. Int. Conf. on Image Analysis and Processing, IEEE Computer Society, Venice, Italy, Sep. 1999: 215 – 220.

[9] 车武军，杨勋年，汪国昭. 动态骨架算法 [J]. 软件学报，2003, 14(4): 818 – 823.