自动修复短时序违反路径的 FPGA 布线算法*

刘 洋[1,2], 杨海钢[1], 喻 伟[1,2], 蔡 刚[1], 徐维涛[1]
(1. 中国科学院电子学研究所 可编程芯片与系统研究室, 北京 100190; 2. 中国科学院大学, 北京 100049)

摘 要: 为了解决寄存器保持时间不满足而引起的短路径问题,提出一种自动修复短时序违反路径的 FPGA 布线算法。在 VPR 时序布线算法整体布线布通之后,调用短路径时序分析来获取违反短时序约束的布线连接,然后通过修改代价函数,对每条违反短时序约束的连接进行增量布线,使每条连接的路径延时尽可能达到满足短时序约束所需的延时。实验结果表明,本算法与 VPR 时序驱动布线算法相比,能够平均修复 94.7% 的短时序违反路径,而运行时间仅增加了 6.8%。

关键词: FPGA; 布线; 短时序违反路径; 代价函数; 增量布线
中图分类号: TP302.1; TP301.6 **文献标志码**: A **文章编号**: 1001-3695(2014)01-0066-04
doi:10.3969/j.issn.1001-3695.2014.01.015

Automatic repairing short-path violations FPGA routing algorithm

LIU Yang[1,2], YANG Hai-gang[1], YU Wei[1,2], CAI Gang[1], XU Wei-tao[1]
(1. *Programmable Chip & System Research Laboratory, Institute of Electronics, Chinese Academy of Sciences, Beijng* 100190, *China*; 2. *University of Chinese Academy of Sciences, Beijing* 100049, *China*)

Abstract: This paper presented a routing algorithm that automatically repaired short-path violations in order to solve short-path problems resulting from holdtime violations in FPGAs. After the execution of VPR timing-driven routing algorithm, the proposed algorithm invoked short-path timing analysis to identify short-path violation connections, and then modified cost function and incrementally rerouted these connections to satisfy short-path constraints. Experimental results demonstrate that the proposed method can repair 94.7% of short-path violations, while, compared with the VPR, the runtime only increases by 6.8% on average.

Key words: FPGA; routing; short-path violations; cost function; incrementally reroute

0 引言

时序是衡量 FPGA 性能的一个关键指标,而布线作为 FPGA 自动化辅助设计流程的重要一环,其结果对电路的时序性能具有关键的作用。FPGA 的时序问题分成长路径和短路径两类问题。长路径问题通常是由于电路中的路径延时过长而引起的时钟频率不满足、输入端口到寄存器建立时间不满足或寄存器到输出端口不满足要求时间等问题;短路径问题则一般是指由于路径延时过短而引起的寄存器到寄存器或输入端口到寄存器保持时间不满足等问题[1]。如果 FPGA 应用电路中存在长路径违反的情况,还可以在较低的时钟频率下运行;但如果存在短路径违反的情况,电路就无法在任何频率下工作。因此 FPGA 应用电路中的短路径优化问题变得尤为重要,研究 FPGA 的短路径优化布线算法具有重要的理论和现实意义。

当前 FPGA 布线算法领域的研究成果大部分集中在长路径时序问题的优化上[2~8],很少有学者去研究 FPGA 的短路径优化问题。早期的 FPGA 短路径优化方法主要采用人工修复的方法[1],在违反短路径约束的路径上人为地增加延时,这种方法随着电路设计规模和时钟结构复杂度的增加而变得非常繁重。后来,Fung 等人[9,10]提出一种同时满足建立时间和保持时间约束的布线算法,通过延时分配把路径的时序约束转换为布线连接的约束,进而改变布线的代价函数来影响布线路径的延时。最近,中国科学院电子所的黄娟等人[11]提出了减少毛刺的低功耗布线算法,通过修改代价函数来使得信号的到达时间基本趋于一致。应用此种方法也能减少时钟偏斜,进而减少寄存器之间的保持时间不满足问题。但上面两种方法都是在布线过程中通过优化布线路径来减少违反短路径约束的情况,而对于布线之后仍然不满足的短时序违反路径则无法进一步优化。

基于以上的研究分析,本文提出了一种布线之后自动修复短时序违反路径的 FPGA 布线算法。该算法首先调用 VPR 时序算法进行整体布线,在整体布线布通之后,通过调用短路径时序分析来获取违反短路径时序约束的布线连接,再对每条违反短路径时序约束的连接进行增量布线。在增量布线时,本算法通过修改代价函数,使得每条连接的路径延时尽可能达到满足时序约束所需要的延时。相对于 VPR 时序驱动布线算法,

收稿日期: 2013-04-17; **修回日期**: 2013-05-27 **基金项目**: 国家科技重大专项资助项目(2013ZX03006004);国家自然科学基金资助项目(61106033)

作者简介:刘洋(1983-),男,河南正阳人,博士研究生,主要研究方向为大规模集成电路设计自动化技术(liuy@mail.ie.ac.cn);杨海钢(1960-),男,上海人,研究员,博导,博士,主要研究方向为高速可编程逻辑芯片设计、数模混合信号 SOC 设计;喻伟(1986-),男,河南周口人,博士研究生,主要研究方向为大规模集成电路设计自动化;蔡刚(1980-),男,湖北麻城人,助理研究员,博士,主要研究方向为可编程逻辑芯片设计;徐维涛(1983-),男,山东威海人,助理研究员,硕士,主要研究方向为大规模集成电路设计自动化.

本算法在增加 6.8% 左右的运行时间的情况下,平均能修复 94.7% 的短时序约束违反的路径,提高了电路的时序性能。

1　背景知识

1.1　FPGA 短路径问题

FPGA 的短路径问题通常是由于路径延时过短导致的寄存器保持时间不满足的问题。在图 1 所示的例子中,寄存器 A 的输出信号经过一段组合路径的延时到达寄存器 B 的输入端口,A 到 B 的路径若要满足 B 的保持时间,必须满足下列关系。

$$T_{\text{fast}}(\text{clk},A)+\text{Micro tco}+T_{\text{fast}}(A,B)\geqslant T_{\text{slow}}(\text{clk},B)+\text{Micro th} \quad (1)$$

其中:$T_{\text{fast}}(\text{clk},A)$、$T_{\text{slow}}(\text{clk},B)$ 分别代表时钟信号 clk 到两个寄存器 A 和 B 时钟输入端口的最快传输延时和最短传输延时;Micro tco 表示寄存器 A 的固有时钟输出延时;Micro th 表示寄存器 B 的固有保持时间延时;$T_{\text{fast}}(A,B)$ 表示信号从源寄存器 A 到终点寄存器 B 的最快传输延时。式(1)决定了信号从 A 到 B 的组合路径延时 $T_{\text{fast}}(A,B)$ 需要满足

$$T_{\text{fast}}(A,B)\geqslant T_{\text{slow}}(\text{clk},B)+\text{Miro th}-T_{\text{fast}}(\text{clk},A)-\text{Miro tco} \quad (2)$$

如果在布线之后 $T_{\text{fast}}(A,B)$ 的延时不能满足式(2),则信号从 A 到 B 的路径就不能满足 B 的保持时间,此路径就是一条短时序违反的路径。

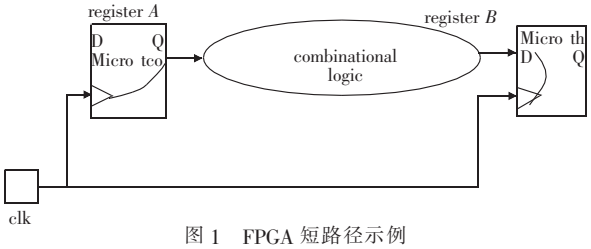

图 1　FPGA 短路径示例

1.2　长路径优化算法

FPGA 长路径优化布线算法中最具代表性的 PathFinder 布线算法[3] 采用了一种拥塞—协商的机制,允许布线过程中出现拥塞。当拥塞发生时,通过调节拥塞资源点的代价,使得最需要拥塞资源的线网最终使用这个资源,而其他线网由于该资源代价大而绕道选择其他资源,从而消除了拥塞。此外,通过在代价函数中引入关键度因子来调节拥塞代价和延时代价的比重,从而在布线中有效地平衡拥塞和时序这两个问题,既布通了电路,又优化了关键路径的延时。VPR 时序驱动算法[5] 在 PathFinder 时序算法的基础上进行了一些改进,使算法的性能更优。VPR 时序驱动布线算法的代价函数为

$$\begin{aligned}\text{cost}(n)=&\text{Crit}(i,j)\cdot\text{delay}(n)+\\&[1-\text{Crit}(i,j)]\times b(n)\times h(n)\times p(n)\end{aligned} \quad (3)$$

其中:$b(n)$ 表示使用该节点的基本代价;$h(n)$ 表示在节点的历史拥塞次数;$p(n)$ 为节点当前被使用的次数;$\text{delay}(n)$ 为路径延时;$\text{Crit}(i,j)$ 表示连接 (i,j) 的关键度,计算式为

$$\text{Crit}(i,j)=\max\left([\text{MaxCrit}-\frac{\text{slack}(i,j)}{D_{\max}}]^{\eta},0\right) \quad (4)$$

式中:$D_{\max}$ 为关键路径延时;η 表示该连接的时延折中系数;MaxCrit 表示拥挤度折中系数;$\text{slack}(i,j)$ 为第 i 条线网源与第 j 个终节点的松弛时间,计算式为

$$\text{slack}(i,j)=T_{\text{required}}(j)-T_{\text{arrival}}(i)-\text{delay}(i,j) \quad (5)$$

其中:$T_{\text{required}}(j)$ 表示到达节点 j 的要求时间;$T_{\text{arrival}}(i)$ 表示节点 i 的到达时间;$\text{delay}(i,j)$ 为节点 i 到 j 的延时。T_{arrival} 和 T_{required} 的计算式分别为

$$T_{\text{arrival}}(i)=\max_{\forall j\in\text{fanin}(i)}\{T_{\text{arrival}}(j)+\text{delay}(j,i)\} \quad (6)$$

$$T_{\text{required}}(i)=\min_{\forall j\in\text{fanout}(i)}\{T_{\text{required}}(j)-\text{delay}(j,i)\} \quad (7)$$

从式(6)和(7)中可以看出,T_{arrival} 和 T_{required} 分别表示节点的最慢到达时间和最快要求时间。

2　本文算法

2.1　算法流程

传统的 VPR 布线算法等虽然在解决 FPGA 布线的长路径优化问题上取得了较好的效果,但其布线过程中并没有考虑短路径优化的问题。文献[10]提出在布线过程中同时优化长路径和短路径问题,但算法不能保证在布线过程中能解决所有的短路径问题;而其以布通为目标的机制,使得在布通之后某些仍然违反短时序约束的路径失去了继续优化的机会,导致布线结果中存在短路径问题,电路无法正常工作。如果算法布通后不立即停止运行,而对其违反短时序约束路径的布线连接重新进行布线,使它们的布线路径延时适当增加,就能减少或消除违反短时序约束的路径。本文提出了一种自动修复短时序约束违反路径的 FPGA 布线算法,算法首先调用 VPR 时序算法进行整体布线,在整体布线布通之后,调用短路径时序分析来获取违反短时序约束的布线连接,然后对每条违反短时序约束的连接进行增量布线。算法在为连接增量布线时,根据短路径松弛时间和迭代次数修改代价函数,增加连接的路径延时,使其尽可能达到满足时序约束所需要的延时。整个算法的流程如图 2 所示。

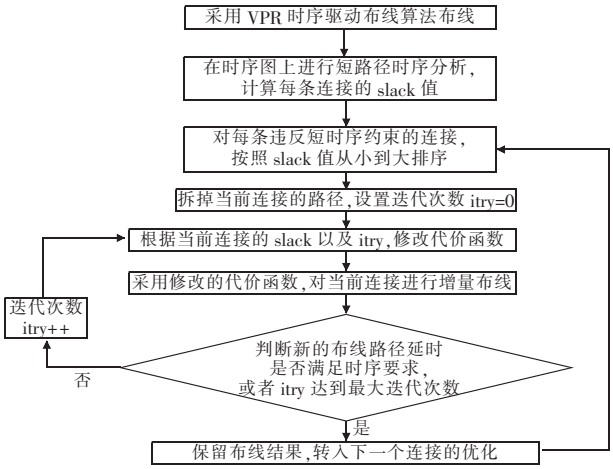

图 2　本文算法流程

主要分为三个步骤:

a)整体布线。采用 VPR 时序驱动布线算法为所有线网进行布线,布线代价函数采用式(1)。

b)短路径分析。在整体布线布通之后,在时序图上进行短路径分析,分析出每个节点的最快到达时间 T_{arrival} 和最慢要求时间 T_{required},进而计算出每条起点—终结点连接的短路径松弛时间 slack。如果某条连接的短路径松弛时间 slack 为负数,表明该连接是一条违反短路径时序约束的布线连接。

c)增量布线。对每条违反短路径时序约束的布线连接,使用余下的布线资源,采用 VPR 时序驱动布线算法进行增量布线。在增量布线时,通过修改代价函数,增加连接的布线路径延时,使其尽可能达到满足时序约束所需要的延时。

2.2 短路径时序分析

时序分析是在时序图 $G\langle V,E\rangle$ 上进行。其中 V 是节点的集合,代表寄存器、查找表等基本逻辑单元的输入、输出端口;E 是边的集合,表示这些节点之间的连接关系。传统的长路径分析在时序图上采用式(6)和(7)的分析方法,计算出时序图上每个节点的最慢到达时间 T_{arrival} 和最快要求时间 T_{required},最后采用式(5)计算出每条起点—终结点连接的长路径松弛时间 slack。而短路径时序分析则不同于长路径时序分析的方法,需要分析出节点的最快到达时间和最慢要求时间,这样才能准确分析出每条路径是否满足短路径时序约束。最快到达时间和最慢要求时间的计算式分别如下:

$$T_{\text{arrival}}(i)=\min_{\forall j\in \text{fanin}(i)}\{T_{\text{arrival}}(j)+\text{delay}(j,i)\} \tag{8}$$

$$T_{\text{required}}(i)=\max_{\forall j\in \text{fanout}}(i)\{T_{\text{required}}(j)-\text{delay}(j,i)\} \tag{9}$$

连接(i,j)的短路径松弛时间计算式为

$$\text{slack}(i,j)=T_{\text{arrival}}(i)-T_{\text{required}}(j)+\text{delay}(i,j) \tag{10}$$

slack <0 表明该连接的延时不足以满足终结点寄存器的保持时间关系,该连接是一条违反短时序约束的连接。

2.3 增量布线

增量布线是指不改变其他绝大部分线网的布线路径,只为某条线网或某条连接进行重新布线,并且重布的时候不占用其他线网的布线路径中已经占用的布线资源点。本文算法对每条违反短路径时序约束的连接进行增量布线,其优点在于没有改变其他线网的布线路径,而只改变需要重布连接的布线路径,因此具有较高的优化成功率和效率。

在本文算法中,增量布线的目的是通过重新布线,延长重布连接的布线路径延时,以达到满足短时序约束所需要的延时。因此需要修改布线的代价函数,使其布线后的路径延时接近于所需的延时。在为某条连接进行增量布线时,因为没有其他线网与其竞争,因此在代价函数中不用考虑拥塞延时,只需考虑延时代价。代价函数如式(11)所示。

$$\text{cost}(n)=\text{delay}(n) \tag{11}$$

每条连接的所需延时可以通过短路径松弛时间 slack 获取到,因此评价当前已布路径所有节点的代价函数 totalCost(n) 设计为如下公式:

$$\text{totalCost}(n)=T_{\text{known}}(n)+\text{cost}(n) \tag{12}$$

$$\text{totalCost}(n)=|1\ \text{ns}\times\text{itry}-\text{slack}(i,j)-\text{totalCost}(n)| \tag{13}$$

其中:$T_{\text{known}}(n)$表示已有路径上节点的代价总和;itry 表示重布的次数;slack(i,j)表示当前连接(i,j)的短路径松弛时间。为某个连接只重布一次,未必能到达所需的延时,因此增量布线时采用多次循环迭代为该连接布线。在每次布线后,评估新的布线延时是否满足要求。如果满足要求,则停止对该连接的优化;否则,增加连接预期的延时,修改代价函数,并重新布线。为每条连接重布的最大迭代次数设置为 δ。当迭代次数超过 δ 时,即使没有满足要求,也停止对该连接的优化。通过实验得出,当 $\delta=10$ 时,就可以基本满足所有的连接重布要求。为一条连接(i,j)增量布线的伪代码为

```
声明: RT(i): 当前处理的线网 i 的节点集合
      Heap:用于存储当前候选节点的堆
      IsValid(n):节点 n 是否可用的资源
      itry:连接重布的次数
  for(每条线网 k,k≠i){
    for(线网 i 布线路径的每个节点 n){
        IsValid(n) = false
    }
    itry = 0;
  while(itry < δ){
    从线网 i 路径中拆掉终结点 j 的布线路径
    将线网已有路径节点加入到 Heap 中
    while(终点 sink(i, j)没有找到){
      移除 Heap 队列首元素 m
      更新 m 的 cost 值
      for(节点 m 的所有扇出节点 n){
        if(IsValid(n) = true){
          根据式(11)~(13)计算节点 n 的 cost 值
          将节点 n 存入 Heap 中
        }//end if
      }//end for
      for(到 sink(i, j)的所有路径节点 n){
        更新路径上节点的延时
        将节点 n 存到 RT(i)
      }//end for
    }//end while
    根据 sink(i, j)新的延时,计算 sink(i, j)的短路径松弛时间 slack
    if(slack >0)//表示已经满足短路径时序
      break;//退出 sink(i,j)的优化
    itry ++ ;
  }//end while
```

3 实验结果

本文实验采用岛型的 FPGA 芯片结构:每个逻辑块包含八个 4 输入的基本逻辑单元,通道采用长度为 4 的双向连接线,连接开关为全连接,交叉开关采用 universal 结构。实验的硬件平台为:CPU 3.3 GHz, 内存 2.0 GB。由于 FPGA 的短路径问题是由于寄存器保持时间不满足引起的问题,因此实验采用 MCNC 测试电路中的 10 个时序电路作为实验电路,这 10 个测试电路的逻辑块数目、I/O 数目及线网数目如表 1 所示。

表 1 测试电路

电路名称	逻辑块数目	I/O 数目	线网数目	电路名称	逻辑块数目	I/O 数目	线网数目
diffeq	189	103	1 033	elliptic	454	245	2 247
dsip	172	426	762	clma	1 055	144	5 354
frisc	446	136	2 022	s298	243	10	710
s38417	802	135	4 410	bigkey	214	426	1 040
tseng	133	174	801	s38584.1	806	342	4 183

对这 10 个测试电路先用 VPR 的装箱算法进行装箱,然后用 VPR 的布局算法进行布局,最后分别调用本文提出的布线算法、VPR 时序布线算法以及文献[10]的布线算法进行布线。为使电路中存在短时序约束违反的路径,时钟信号不作为全局信号,也需要经过布线,这样寄存器之间就存在由于时钟信号传输延时不一致导致的时钟偏斜,从而产生保持时间不满足的路径。在每种布线算法布线之后,对短时序约束违反路径个数进行统计,统计结果如表 2 所示。从表 2 中可以看出,本文算法与 VPR 时序驱动布线算法相比,短时序约束违反路径个数最大减少了 100.0%,最小减少了 88.2%,平均减少 94.7%。而与文献[10]的算法相比,本文算法的短时序约束违反路径个数也平均减少了 91.7%。结果显示,本文算法能修复绝大部分的短时序违反路径。

由于本文算法在 VPR 时序驱动布线算法完成之后再逐步对目标连接进行优化,所以会增加一些运行时间。但增加的运行时间主要是用于对有限的布线连接进行少量次数的重新布线,因此增加的运行时间相对于整个算法的运行时间只占很少的一部分。表 3 列出了对 10 组 MCNC 电路布线时间的比较结

果。从表 3 中可以看出,本文算法与 VPR 时序驱动布线算法相比,运行时间增加比例最大的是 diffeq 电路,增加了 13.7%;比例最小的是 clma 电路,增加了 0.4%;运行时间平均增加了 6.8%。从运行时间的比较结果中可看出,修复的时序违反路径越多,增加的时间比例也就越大。实验证明,本文算法只需要少量的运行时间,就可以达到修复短时序违反路径的目的。

表 2　短时序约束违反路径个数的比较

电路	VPR 时序布线	文献[10]	本文算法		
			违反路径个数	比 VPR 减少/%	比文献[10]减少/%
diffeq	37	26	2	94.6	92.3
dsip	5	7	0	100.0	100.0
frisc	17	12	2	88.2	83.3
s38417	7	3	0	100.0	100.0
tseng	2	1	0	100.0	100.0
elliptic	8	6	0	100.0	100.0
clma	2	1	0	100.0	100.0
s298	3	1	0	100.0	100.0
bigkey	4	1	0	100.0	100.0
s38584.1	9	2	1	88.9	50.0
平均	9.4	6	0.5	94.7	91.7

表 3　运行时间的比较

电路	VPR 时序布线算法/ms	本文算法/ms	比 VPR 增加%	电路	VPR 时序布线算法/ms	本文算法/ms	比 VPR 增加%
diffeq	3 766	516	13.7	clma	26 656	109	0.4
dsip	2 125	63	3.0	s298	3 156	16	0.5
frisc	8 438	875	10.4	bigkey	2 953	78	2.6
s38417	11 391	1 500	13.2	s38584.1	10 516	1 266	12.0
tseng	1 688	94	5.6	平均	7 931	543	6.8
elliptic	8 625	922	10.7				

4　结束语

本文提出一种自动修复短时序违反路径的 FPGA 布线算法。该算法首先调用 VPR 时序算法进行整体布线,在整体布线布通之后,调用短路径时序分析来获取违反短路径时序约束的布线连接,然后对每条违反短路径时序约束的连接进行增量布线。在增量布线时通过修改代价函数,使得每条连接的路径延时尽可能达到满足时序约束所需要的延时。实验结果表明,该算法与 VPR 时序布线算法相比,在增加少量运行时间的情况下,能够修复绝大部分的短时序约束违反路径。

参考文献:

[1] SHENOY N, BRAYTON R, SANGIOVANNI-VINCENTELLI A. Minimum padding to satisfy short path constraints[C]//Proc of IEEE/ACM International Conference on Computer-Aided Design. Santa Clara: ACM Press, 1993: 156-161.

[2] FRANKLE J. Iterative and adaptive slack allocation for performance-driven layout and FPGA routing [C]//Proc of the 29th ACM/IEEE Design Automation Conference. Los Alamitos: IEEE Computer Society, 1992: 536-542.

[3] McMURCHIE L, EBELING C. PathFinder: a negotiation-based performance-driven router for FPGAs [C]// Proc of the 3rd International ACM Symposium on Field-Programmable Gate Arrays. New York: ACM Press, 1995: 111-117.

[4] MLEXANDER M, ROBINS G. New performance-driven FPGA routing algorithms[J]. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems, 1996, 15(12): 1505-1517.

[5] BETZ V, ROSE J. VPR: a new packing, placement and routing tool for FPGA research [C]//Proc of the 7th International Workshop on Field Programmable Logic and Application. London: Springer, 1997: 213-222.

[6] BETZ V, ROSE J, MARQUARDT A. Architecture and CAD for deep-submicron FPGAs [M]. Norwell: Kluwer Academic Publishers, 1999.

[7] RUBIN R, De HON A. Timing-driven pathfinder pathology and remediation: quantifying and reducing delay noise in VPR-pathfinder [C]//Proc of the 19th International ACM Symposium on Field-Programmable Gate Arrays. New York: ACM Press, 2011: 173-176.

[8] ROSE J, LUU J, YU Chi-wai, *et al*. The VTR project: architecture and CAD for FPGAs from verilog to routing[C]//Proc of the 20th International ACM Symposium on Field-Programmable Gate Arrays. New York: ACM Press, 2012: 77-86.

[9] FUNG R, BETZ V, CHOW W. Simultaneous short-path and long-path timing optimization for FPGAs[C]//Proc of IEEE/ACM International Conference on Computer-Aided Design. Washington DC: IEEE Computer Society, 2004: 838-845.

[10] FUNG R, BETZ V, CHOW W. Slack allocation and routing to improve FPGA timing while repairing short-path violations[J]. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems, 2008, 27(4): 686-697.

[11] 黄娟,杨海钢,李威,等. 可编程逻辑阵列减少毛刺的低功耗布线算法[J]. 计算机辅助设计与图形学报, 2010, 22(10): 1664-1670.

(上接第 65 页)

[13] ANGIULLI F. Fast nearest neighbor condensation for large datasets classification [J]. IEEE Trans on Knowledge and Data Engineering, 2007, 19(11): 1450-1464.

[14] FAYED H A, ATIYA A F. A novel template reduction approach for the K-nearest neighbor method [J]. IEEE Trans on Neural Networks, 2009, 20(5): 890-896.

[15] NIKOLAIDIS K, GOULERMAS J Y, WU Q H. A class boundary preserving algorithm for data condensation [J]. Pattern Recognition, 2011, 44(3): 704-715.

[16] De HARO-GARCIA A, GARCÍA-PEDRAJAS, Del CASTILLO A R. Large scale instance selection by means of federal instance selection [J]. Data & Knowledge Engineering, 2012, 75: 58-77.

[17] CÉSAR G O, De AIDA H G, NICOLÁS G P. Democratic instance selection: a linear complexity instance selection algorithm based on classifier ensemble concepts [J]. Artificial Intelligence, 2010, 174(5-6): 410-441.

[18] SPECHT D F. Probabilistic neural networks [J]. Neural Networks, 1990, 3(1): 109-118.

[19] DUDA R O, HART P E, STORK D G. Pattern classification[M]. 2nd ed. Beijing: China Machine Press, 2005.

[20] 叶志锋,孙健国. 基于概率神经网络的发动机故障诊断[J]. 航空学报, 2002, 23(2): 155-157.

[21] 杨鼎强,肖淑苹,蒋加伏. 基于差异演化概率神经网络的纹理图像识别[J]. 计算机工程与应用, 2008, 44(11): 179-181, 198.

[22] 程智辉,陈将宏. 基于概率神经网络的乳腺癌计算机辅助诊断[J]. 计算机仿真, 2012, 29(9): 166-169.

[23] SEUNG H S, OPPER M, SOMPOLINSKY H. Query by committee [C]//Proc of ACM Workshop on Computational Learning Theory. 1992: 287-294.

[24] FRANK A, ASUNCION A. UCI machine learning repository [EB/OL]. (2010). http://archive.ics.uci.edu/ml.

[25] VERIKAS A, LIPNICKAS A, MALMQVIST K, *et al*. Soft combination of neural classiers: a comparative study [J]. Pattern Recognition Letters, 1999, 20(4): 429-444.