

一种基于 CAN-tree 快速构建算法

陈 刚¹, 闫英战¹, 刘秉权²

(1 广东科技学院 计算机系, 广东 东莞 523083; 2 哈尔滨工业大学 计算机科学与技术学院, 黑龙江 哈尔滨 150001)

摘 要: 增量关联挖掘领域是近几年数据挖掘研究的一个热点, 而 CAN-tree 的提出是增量关联挖掘发展的一个重要里程碑. 虽然 CAN-tree 适用于增量关联挖掘, 但 CAN-tree 构建时间长, 挖掘效率偏低; 针对此问题, 提出了一种基于 CAN-tree 快速构建算法, 该算法增加基于哈希表的辅助存储结构, 用于减少项目的查找时间, 从而提高 CAN-tree 的构建速度; 增加叶子地址表并修改节点结构及改 child 指针为 parent 指针, 目的是减少剪枝时间和条件模式基的生成时间, 从而提高数据挖掘效率. 实验结果表明该算法有效.

关键词: 关联规则; 频繁模式; CAN-tree; 条件模式

中图分类号: TN402

文献标识码: A

文章编号: 1000-7180(2014)01-0076-07

A Fast Construction Algorithm Based on CAN-Tree

CHEN Gang¹, YAN Ying-zhan¹, LIU Bing-quan²

(1 Department of Computer, Guangdong University of Science & Technology, Dongguan 523083, China;

2 School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

Abstract: The incremental association rules mining is a hot point for data mining research in recent years, and the appearance of CAN-tree is an important milepost in development of incremental mining association. Although the CAN-tree is suitable for incremental association rules mining, the CAN-tree needs long construction time, then causes low mining efficiency; In order to solve this problem, a fast construction algorithm based on CAN-tree is presented, the algorithm adds the auxiliary storage structure based on Hash which can be used to reduce project-search time, so that the construction of CAN-tree speed can be improved; Then the algorithm adds leaves address sheet and modifies node structure and changes child pointer to parent pointer, which reduce the generation time of conditional pattern base, so that the efficiency of data mining can be improved. experiment results show that the algorithm is effective.

Key words: association rules; frequent pattern; CAN-tree; conditional mode

1 引言

关联规则是数据挖掘研究的一个重要领域, 而增量关联规则是当今关联规则领域的一个研究热点. 而数据流挖掘则是增量关联规则应用的一个重要领域, 传统的解决方案是对更新后的数据库进行重执行, 但是该方法至少付出重新扫描全局事务数据库的高昂代价, 为了解决增量关联规则问题, Cheung 等人提出了增量挖掘的思想^[1], 即在充分利

用原有的挖掘基础, 花费较小的代价就可以得到数据库更新后的关联规则. 如早期的基于 Apriori 算法的 FUP2^[2] 及 WEP 算法^[3] 等.

为了克服 Apriori 算法需要多次扫描全局数据库而产生大量的侯选项集的局限性, Han Jiawei 提出了 FP-Growth 思想, 并以该思想为基础构建了频繁模式树 (Frequent Pattern tree), 该算法大大改进了 Apriori 算法挖掘频繁项集需多次扫描数据库的缺陷. 但是该算法及其改进算法如李等提出的基于

收稿日期: 2013-03-18; 修回日期: 2013-05-02

基金项目: 国家自然科学基金项目 (61073127); 东莞市科技计划项目 (2012108102030)

改进的 FP-tree 的频繁模式挖掘算法^[4]均需要扫描两次数据库,陈等提出的基于 FP-tree 的快速构建算法^[5],马丽生等提出的基于 FP-tree 的最大频繁项目集挖掘算法^[6],姜晗等提出的基于 FP-tree 快速挖掘频繁项集的算法^[7],这些改进算法虽然进行各种改进,效率有一定提高,但是随着事务数据不断增加或者当最小支持度发生变化时,FP-tree 要重新构建,这对大数据时代的到来是致命的,因此 FP-tree 不适合于增量挖掘.后来 Leung 等对 FP-tree 进行改进,提出了不随着数据量增加而重新调整项头表的算法,即基于自然序树 CAN-Tree(canonical-order tree)结构构建算法^[8],该算法克服了 FP-tree 构建算法的缺陷,仅需要一次构建,就可多次挖掘.但由于 CAN-tree 中保留了所有非频繁信息,因此 CAN-tree 构建速度较慢,且导致后期的生成条件模式基进行数据挖掘的效率更低.

鉴于此,本文提出一种基于 CAN-tree 快速构建算法 A fast construction algorithm based on CAN-tree (FCCAN),该算法对 CAN-tree 结构做了如下修改:(1)增加了哈希存储表,提高查询速度;(2)修改了链接指针,由父到子改为由子到父并且增加叶子头表,提高条件模式基的生成速度;(3)修改节点结构,提高了剪枝速度.实验结果表明,该算法有效.

2 CAN-tree 构建算法的相关理论

在增量挖掘领域,主要是基于 Aprior 的增量挖掘算法及基于 FP-tree 的频繁模式增长算法,及基于自然序树(CAN-tree)的挖掘算法.但是由于 Aprior 算法会产生大量的候选频繁项集且全局数据库的多次扫描会给系统带来很大开销;而频繁模式增长算法(FP 增长算法)使用一种称为 FP-tree 的树型结构,且在 FP-tree 的构建过程中采用分而治之的策略,因此得到全部频繁项集的同时但不需要产生大量候选频繁项集,关于 FP-tree 的构建及 Aprior 算法的相关理论可以参考有关文献^[9].

虽然 FP-tree 结构可以实现增量数据动态挖掘,但是由于 FP-tree 在构建过程中要两次扫描全局事务数据库,并且动态地向 FP-tree 添加新数据或者改变最小支持度时,FP-tree 要进行重构,这些都将消耗大量的 CPU 时间,从而影响挖掘的效率.

后来有学者提出了只要扫描两次全局数据库且项头表不随数据的添加而改变的基于自然排序树的 CAN-tree 结构.

2.1 CAN-tree 的相关定义

CAN-tree 结构与 FP-tree 结构有些类似,仅仅是在于其构建基础. CAN-tree 在构建项头表时按一个固定顺序(字典、字母等)进行排列, CAN-tree 结构的如下:

(1)定义一个根节点,其值为 null.

(2)作为根节点的子树节点集合,其子树中的每个节点包括项目名 item、节点计数器 count、和指向孩子指针 child[]及指向同名节点指针 back,与 FP-tree 结构不同在于,FP-tree 只有一棵子树,而 CAN-tree 可能有多棵子树.节点设计与 FP 基本相同.

(3)项头表(head_table),其包括一个频繁项标识 item、一个指向该树具有该标识第一个频繁节点的指针 list,

该项头表不同于 FP-tree,FP-tree 是按照支持度降序排列的,而 CAN-tree 是按指定顺序排列,在增量数据添加时也不变更顺序.

2.2 CAN-tree 构建过程

(1)获得已经按某种设计顺序(字典顺序、字母顺序)排列的固定项头表,并以此创建根节点,且用 null 标记之;

(2)将扫描数据库得到的所有事务按固定项头表的顺序排序,并且对事务中的每个项 x 进行插入操作,操作时遵循第(3)步;

(3)依次遍历与 x 同名节点路径,若 x 项对应的已经建立的同名节点的父节点与执行插入的事务的中项 x 的前项名相同,则将与 x 项同名的节点的计数增加,否则创建一个新的节点 N_1 ,新节点 N_1 的父节点应该与插入事务中的 x 项的前项名相同;依次完成(3),直到所有的事务数据全部插入完成,此时 CAN-tree 构建完成.

CAN-tree 结构优越在于:项头表中的数据项排序不受数据更新的影响以及树中的节点的支持度计数不小于其子节点的支持计数的总和,因此只要给定顺序的项头表 CAN-tree 的结构是唯一的,并不会因为数据增加或者支持度的改变而调整树的结构, CAN-tree 良好的结构说明其构成的树对增量挖掘具有很好的适应性,因此只有 CAN-tree 结构树才真正的实现了“一次构建、多次挖掘”.

3 基于 CAN-tree 快速构建算法 FCCAN

虽然 CAN-tree 具有良好的性质,但是从 CAN-tree 结构分析可知, CAN-tree 结构也有以下不足:

针及叶子地址表 $\text{leaf}[k]$ 与叶子之间互指指针用虚线表示,并且增加一个 head 结构和 leaf 结构,其中 head 用于记录该项所在节点的 CAN-tree 结构树的子树的根节点地址, left 用于记录当前节点是否为叶子,当为叶子时, leaf 记录了指向叶子地址表的地址,当不为叶子时, leaf 的值为空,如图 2 所示。

对比图 1 和图 2,可以看出改进的 CAN-tree 结构比传统的 CAN-tree 结构多一个叶子地址表,并且树的链接方式由父—>子的指针改为由子—>父,这样做的目的是能快速地生成条件模式基,并且在新结构树的节点中增加了两项,即用于存储结构树中子树根节点地址的 head 项和区分该节点是否为叶子的 leaf 项,这些新增的结构有助快速实现 CAN-tree 结构树的剪枝,由于在新结构树中有 parent 指针,因此新结构树中不需要 null 值的根节点。

3.3 构造新型 CAN-tree 结构

构建新型的 CAN-tree,仍然需要两次数据库,因为在构建 CAN-tree 结构树前要先构建前项头表(基于哈希结构的辅助存储表)和项头表及计算出项目的支持度,所以第一次扫描数据库是为了构建项头表及前项头表,把所有对项头表的访问转到对前项头表的访问。具体实现过程如下:

第一次扫描数据库,获取各项目及项目的支持计数,将所有的项目按固定顺序排列并初始化 T.head,其中 T.head[x]中存放了 x 项的支持计数 count 和树中的第一个 x 项的链节信息 list,同时确定相应的哈希函数并且构建前项头表。第二次扫描数据库则将所有库中的事务依次插入到 CAN-tree 中,构建 CAN-tree 的过程是将 n 个相同事务数据 tran 插入到树中的过程,由于新型 CAN-tree 的结构发生变化,因此相应的插入操作也改变了。对于要插入的事务数据 tran 中的每个数据项 x,若要插入 x 必须记录好 x 的前一数项在所树中的节点 fr,在插入时,若 x 为 tran 中的第一个数据项,则 $\text{fr} = \text{null}$ 。且 T.head[x].count 执行加 n 操作,如果在 T.head[x].list 上存在一个这样的节点 N,使 $N.\text{parent} = \text{fr}$,则 $N.\text{count} = N.\text{count} + n$;否则创建一个新节点 N_1 , $N_1.\text{count} = n$,且 $N_1.\text{parent} = \text{fr}$;若 $\text{fr} = \text{null}$,则 $N_1.\text{leaf}$ 中存放了指向叶子地址表中 $\text{leaf}[k]$ 的地址,且 $\text{leaf}[k]$ 中存放了 N_1 叶子节点地址。否则判断的 N_1 父节点 fr 插入前是否为叶子,若为叶子则执行链接到叶子节点操作,否则执行链接到非叶子节点操作;此时在 $N_1.\text{head}$ 存放了当前 CAN-tree 结构子树根节点地址,该标志由于后续的

剪枝操作。此时应将 N_1 链接树中的与 N_1 具有同名节点的 back 指针中去。最后将 fr 值设置 N(若是创建了新节点,则值为 N_1),为下一个插入做准备。算法流程图如图 3 所示。

其中流程图中的第一项目 (x, n) 表示具有 n 个相同的 x 项目名, p 为节点变量, N, N_1 , 分别表示符合条件的老节点和刚创建的新节点, k 用于统计叶子的数目。

3.4 CAN-tree 剪枝

由于 CAN-tree 在构建过程中不需要支持度 cup,因此生成的 CAN-tree 中包含了所有的信息,即存放了支持度小于 cup 非频繁项目,所以在构建条件模式树时需要使用 cup 参数删除树中所有的非频繁项,由于在 CAN-tree 使用了 parent 指针及叶子地址表,所以很容易得到所有的条件模式基且在不用访问所有节点的情况下就实现了非频繁项的剪枝操作。本文由于实现需要定义一个公式 corre 表示相关度,即具有相同子树根节点相同项目的节点总数,如图 2 其子树根节点分别为 a:3, c:2, d:1, 节点相关度 corre 实现如式(1)所示:

$$\text{corre} = \sum_{\text{back}=\text{T.head, list}}^{\text{back}=\text{null}} N.\text{count}(\text{相同子树根节点}) \quad (1)$$

节点相关度 corre 和支持计数 count 共同用于实现剪枝操作,具体的剪枝过程为,首先从叶子地址表 $\text{leaf}[]$ 中取出第一个叶子节点 p_i ,判断其 $p_i.\text{count} < \text{cup} \& \text{corre} < \text{cup}$,若满足条件则删除该节点,沿着 parent 指针执行,直到 $p_i.\text{count} \geq \text{cup}$ 为止,接着取第二个叶子节点,直到 $k \geq \text{length}(\text{leaf}[])$ 为止,该剪枝的优点在于不需要访问所有的 CAN-tree 中的所有节点,这点利用了 CAN-tree 中节点的支持度计数不小于其子节点的支持计数的总和特征,并且在程序没有使用递归程序,从而快速的实现剪枝,有利于频繁模式树的生成,从而提高挖掘效率,其算法的详细实现流程图如图 4 所示,

其中 p_i 表示当前节点, s 表示 p_i 的孩子节点, y 表示沿着 back 指针查询的节点, t 表示 y 的前一个节点。

4 实验结果分析

实验基于 Intel 3.0 GHz 的双核 CPU, 2 GB 内存, OS 为 Windows XP Professional SP2 的环境, 算法实现工具 Borland C++ Builder 6.0. 实验数据合成数据 T10I4D100K^[10]. 实验从两方面进行,一方

面测试测试增加前项头表(基于哈希表的辅助存储结构)对 CAN-tree 结构树的构建的影响,如图 5 所示;另一方面测试基于新型 CAN-tree 结构对挖掘

效率的影响. 两个实验均在固定的数据集(依次全部加载)和固定支持度为 0.1 情况下实验.

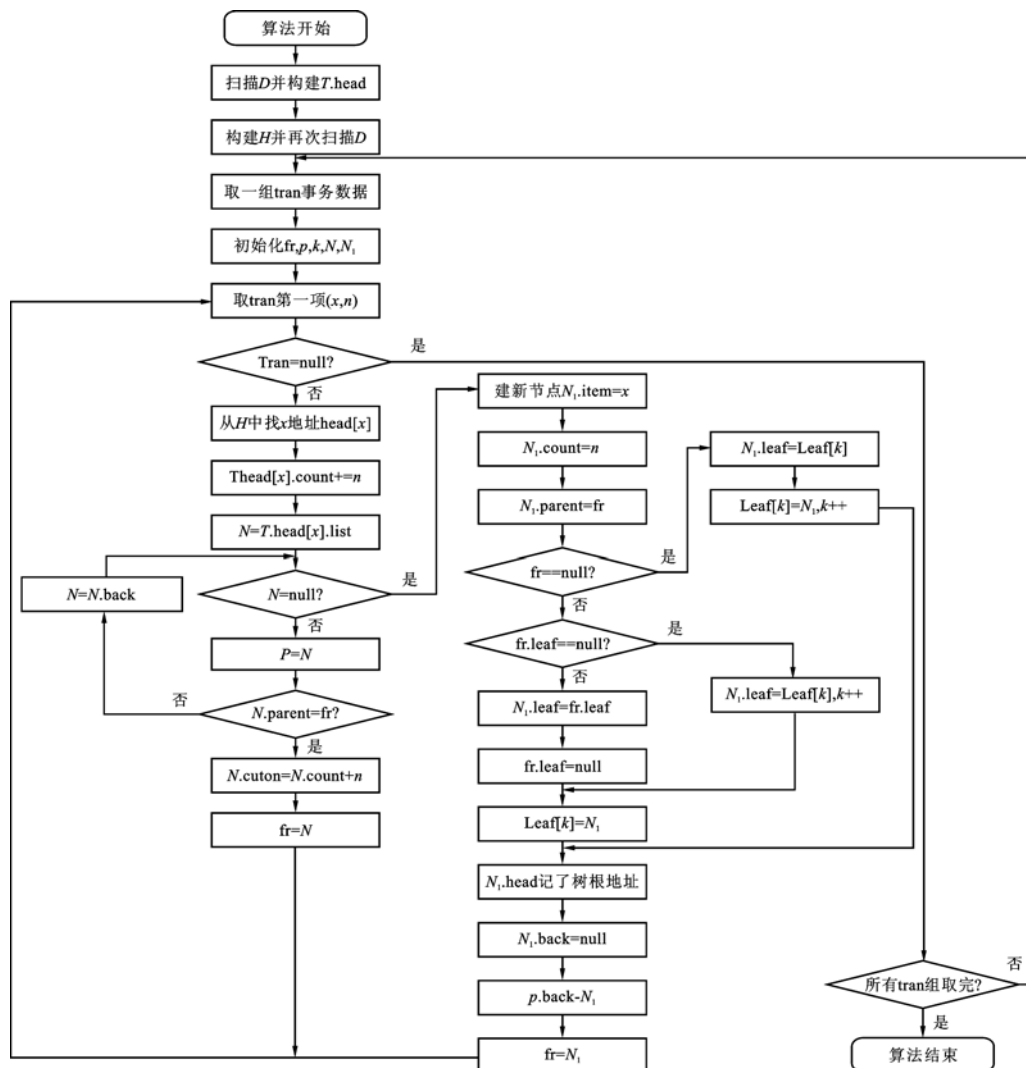


图3 FCCAN 算法实现流程图

从图 5 的测试结果可以看出,由于 FP-tree 在数据增加时,项头表要进行大量排序工作,所以花费时间较 CAN-tree 多,又因为 CAN-tree 中保留了大量非频繁项目,增加大量新节点,也需花费部分时间,因此 FP-tree 构建时间和 CAN-tree 构建时间在某段时期会靠近;而 FCCAN 构建算法由于使用了哈希辅助存储结构,随着数据增加,其算法建树时间比 CAN 构建算法建树的时间少的多,优势越明显.从图 6 的测试结果来看,在 FCCAN 算法构建树上的数据挖掘时间比在 FP 算法构建树上的挖掘时间要少,其原因 FP 构建算法及 CAN 构建算法查找条件模式基是从项头表中逐项查找的,而 FCCAN 算法从叶子地址表开始查找,每条路径只要查找一次,从而比 FP 算法与 CAN 算法节省了重复遍历子模

式所需的时间;当数据量逐渐增大时,条件模式越长,效果越明显.从图 6 中也可看出,FP 构建算法优于 CAN 构建算法,其原因在于 CAN-tree 比 FP-tree 节点多,路径多,所以花费时间相对较多.

5 结束语

本文提出了一种基于 CAN-tree 快速构建算法,该算法对 CAN-tree 的项目头表存储方式进行了改进,增加了一个基于哈希函数辅助存储结构,并对节点结构及节点间的链接指针进行了改进且增加了一个叶子地址表,虽然该算法增加了一定存储开销,但是其显著地减少了 CAN-tree 的构建时间和基于新型 CAN-tree 数据挖掘时间,构建效率和数据挖掘效率均有所提高.实验结果表明,该算法是有

效的.

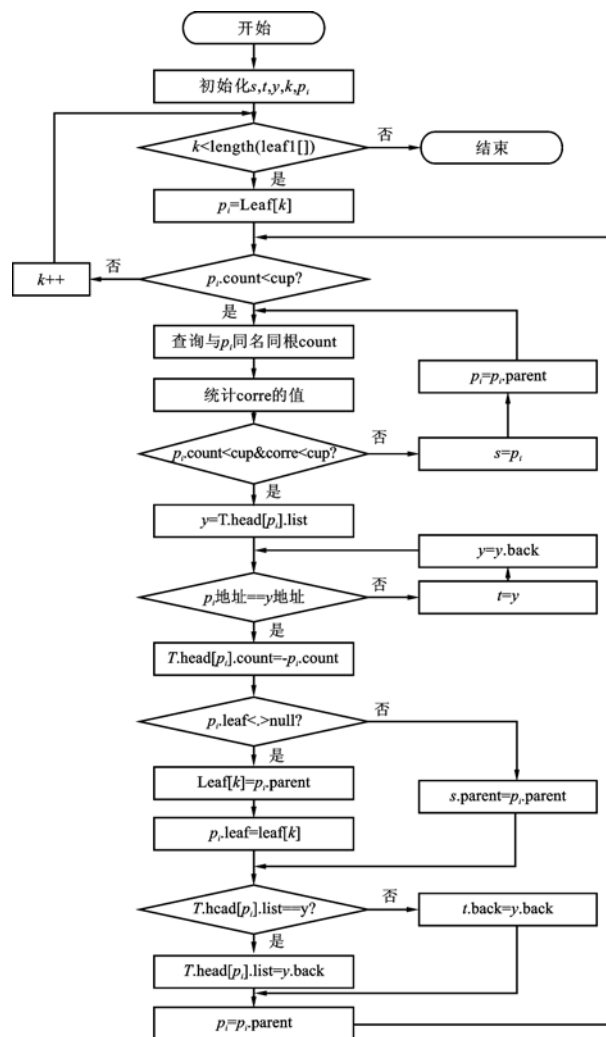


图 4 剪枝程序

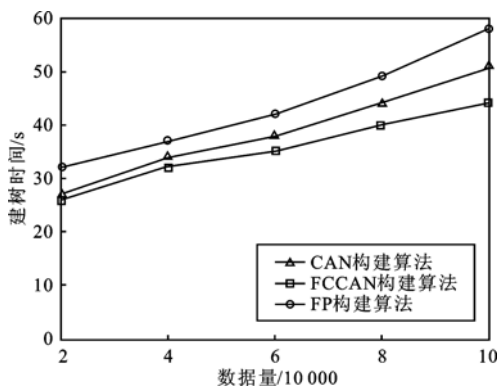


图 5 模式树的构建时间比较

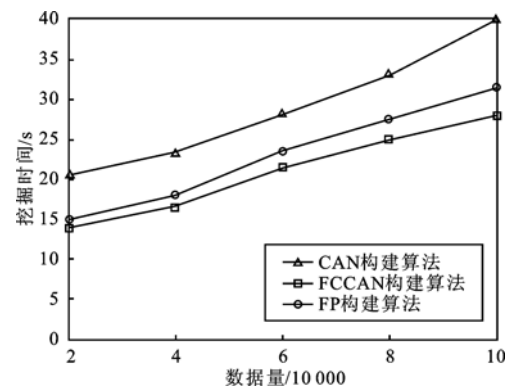


图 6 挖掘时间比较

参考文献:

- [1] Cheung D, Han J, Ng V, et al. Maintenance of discovered association rules in large databases: an incremental updating technique[C]// Proceedings of the 12th International Conference on Data Engineering (ICDE). New Orleans, 1996: 106-114.

- [2] 王云岚, 李增智, 屈科文. 基于候选项集个数上阶的增量式关联规则更新算法[J]. 电子学报, 2004, 32(5): 731-734.
- [3] Ayan N F, Tansel A U, Arkun M E. An efficient algorithm to update large itemsets with early pruning [C]// Proceedings of the 5th ACM International Conference on Knowledge Discovery and Data Mining

- (KDD). USA: San Diego, 1999: 287-291.
- [4] 李也白, 唐辉, 张淳, 等. 基于改进的 FP-tree 的频繁模式挖掘算法[J]. 计算机应用, 2011, 31(1): 101-103.
- [5] 陈治平, 谭义红, 李学勇, 等. 基于 FP-tree 的快速构建算法[J]. 计算机应用, 2011, 31(2): 438-440.
- [6] 马丽生, 邓辉文, 齐逸. 基于 FP-tree 的最大频繁项目集挖掘算法[J]. 计算机工程与设计, 2008, 29(2): 385-388.
- [7] 姜晗, 范建淑, 贾洞. 基于 FP-Tree 快速挖掘频繁项集[J]. 计算机应用与软件, 2010, 27(10): 128-130.
- [8] Leung Carson Kai-Sang, Khan Quamrul I, Li Zhan, et al. CanTree: a canonical-order tree for incremental pattern mining [J]. Knowledge Information System, 2007, 11(3): 287-311.
- [9] 冯洁, 陶宏才. 快速挖掘最大频繁项集[J]. 微电子学与计算机, 2007, 24(5): 123-126.

- [10] Frequent itemset mining implementations repository [EB/OL]. [2010-01-25]. <http://fimi.cs.helsinki.fi>

作者简介:



陈刚 男, (1977-), 硕士, 讲师. 研究方向为数据挖掘、人工智能.

闫英战 男, (1972-), 硕士, 副教授. 研究方向为智能网络、数据挖掘.

刘秉权 男, (1970-), 博士, 副教授. 研究方向为自然语言处理、人工智能.

(上接第 75 页)

- [6] 吴玲, 孙永荣, 俞晓磊, 等. GPS/GLONASS/GALILEO 多星座组合导航系统研究[J]. 中国空间科学技术, 2009(3): 23-28.
- [7] 蔡昌盛, 戴吾蛟, 匡翠林. GPS/GLONASS 组合系统的 PDOP 计算和分析[J]. 测绘通报, 2011(11): 5-7.
- [8] 陈俊平, 吴斌, 胡小工, 等. GPS/GLONASS 时差监测及其在多模定位中的应用[C]//第三届中国卫星导航学术年会-S02 卫星导航信号体制及兼容与互操作. 广州, 2012.
- [9] 谢钢. GPS 原理与接收机设计[M]. 北京: 电子工业出版社, 2009.

作者简介:

翟昆朋 男, (1986-), 博士. 研究方向为 GNSS 接收机导航算法和信号处理算法.

徐建华 男, (1963-), 研究员, 博士生导师. 研究方向为嵌入式软件、无线通讯系统、定位系统及算法研究.

叶甜春 男, (1965-), 研究员, 博士生导师. 研究方向为集成电路设计与应用、纳米加工与制造技术、新型器件.